NEW AGE

FUZZY LOGIC AND NEURAL NETWORKS

BASIC CONCEPTS & APPLICATIONS

Chennakesava R. Alavala

NEW AGE INTERNATIONAL PUBLISHERS

SHOUT
SHARE
IT

FUZZY LOGIC
AND
NEURAL NETWORKS
BASIC CONCEPTS & APPLICATIONS

This page intentionally left blank

FUZZY LOGIC AND NEURAL NETWORKS

BASIC CONCEPTS & APPLICATIONS

Chennakesava R. Alavala
B.E., M.E. (mech.), M.Tech (CAD/CAM)
Ph.D. (mech.), Ph.D. (CAD/CAM)
M.ISTE., M.IIF., M.IIW., M.ISME., M.IE., M.IIPE., C.Engg.

Professor of Mechanical Engineering
Jawaharlal Nehru Technological University
Hyderabad, Andhra Pradesh
India

PUBLISHING FOR ONE WORLD

NEW AGE INTERNATIONAL (P) LIMITED, PUBLISHERS

New Delhi • Bangalore • Chennai • Cochin • Guwahati • Hyderabad
Jalandhar • Kolkata • Lucknow • Mumbai • Ranchi
Visit us at **www.newagepublishers.com**

This page intentionally left blank

To
My Parents

This page intentionally left blank

Preface

This book attempts to provide the reader with basic concepts and engineering applications of Fuzzy Logic and Neural Networks. Back in 2000, the absence of any state-of-the-art (Indian Domain) textbook forced me to write this book.

Some of the material in this book contains timely material and thus may heavily change throughout the ages. The choice of describing engineering applications coincides with the Fuzzy Logic and Neural Network research interests of the readers.

Modeling and control of dynamic systems belong to the fields in which fuzzy set techniques have received considerable attention, not only from the scientific community but also from industry. Many systems are not amenable to conventional modeling approaches due to the lack of precise, formal knowledge about the system, due to strongly non-linear behaviour, due to the high degree of uncertainty, or due to the time varying characteristics. Fuzzy modeling along with other related techniques such as neural networks have been recognized as powerful tools, which can facilitate the effective development of models.

The approach adopted in this book aims at the development of transparent rule-based fuzzy models which can accurately predict the quantities of interest, and at the same time provide insight into the system that generated the data. Attention is paid to the selection of appropriate model structures in terms of the dynamic properties, as well as the internal structure of the fuzzy rules.

The field of neural networks has a history of some five decades but has found solid application only in the past fifteen years, and the field is still developing rapidly. Thus, it is distinctly different from the fields of control systems or optimization where the terminology, basic mathematics, and design procedures have been firmly established and applied for many years. Neural networks are useful for industry, education and research. This book is intended to cover widely primarily the topics on neural computing, neural modeling, neural learning, and neural memory.

Recently, a great deal of research activity has focused on the development of methods to build or update fuzzy models from numerical data. Most approaches are based on neuro-fuzzy systems, which exploit the functional similarity between fuzzy reasoning systems and neural networks. This combination of fuzzy systems and neural networks enables a more effective use of optimization techniques for building fuzzy systems, especially with regard to their approximation accuracy. Neuro-fuzzy models can be regarded as black-box models, which provide little insight to help understand the underlying process.

The orientation of the book is towards methodologies that in the author's experience proved to be practically useful. The presentation reflects theoretical and practical issues in a balanced way, aiming at

readership from the academic world and also from industrial practice. Examples are given throughout the text and six selected real-world applications are presented in detail.

Chennakesava R. Alavala

Contents

Preface *vi*

Chapter 1: Introduction **1-5**

1.1 Fuzzy Logic (FL) 1
1.2 Neural Networks (NN) 2
1.3 Similarities and Dissimilarities Between FL and NN 4
1.4 Applications 4
Question Bank 4
References 5

Part I: Fuzzy Logic

Chapter 2: Fuzzy Sets and Fuzzy Logic **6-18**

2.1 Introduction 6
2.2 What is Fuzzy Logic? 6
2.3 Historical Background 8
2.4 Characteristics of Fuzzy Logic 9
2.5 Characteristics of Fuzzy Systems 9
2.6 Fuzzy Sets 9
2.6.1 Fuzzy Set 9
2.6.2 Support 11
2.6.3 Normal Fuzzy Set 11
2.6.4 α-Cut 11
2.6.5 Convex Fuzzy Set 11
2.6.6 Fuzzy Number 12
2.6.7 Quasi Fuzzy Number 12
2.6.8 Triangular Fuzzy Number 13
2.6.9 Trapezoidal Fuzzy Number 14
2.6.10 Subsethood 14
2.6.11 Equality of Fuzzy Sets 15
2.6.12 Empty Fuzzy Set 15

2.6.13 Universal Fuzzy Set 15
2.6.14 Fuzzy Point 15
2.7 Operations on Fuzzy Sets 16
2.7.1 Intersection 16
2.7.2 Union 16
2.7.3 Complement 17
Question Bank *17*
References *18*

Chapter 3: Fuzzy Relations **19-28**
3.1 Introduction 19
3.2 Fuzzy Relations 19
3.2.1 Classical N-Array Relation 19
3.2.2 Reflexivity 20
3.2.3 Anti-Reflexivity 20
3.2.4 Symmetricity 20
3.2.5 Anti-Symmetricity 20
3.2.6 Transitivity 20
3.2.7 Equivalence 20
3.2.8 Partial Order 20
3.2.9 Total Order 20
3.2.10 Binary Fuzzy Relation 21
3.3 Operations on Fuzzy Relations 21
3.3.1 Intersection 21
3.3.2 Union 22
3.3.3 Projection 23
3.3.4 Cartesian Product of Two Fuzzy Sets 24
3.3.5 Shadow of Fuzzy Relation 24
3.3.6 Sup-Min Composition of Fuzzy Relations 26
Question Bank *27*
References *27*

Chapter 4: Fuzzy Implications **29-40**
4.1 Introduction 29
4.2 Fuzzy Implications 30
4.3 Modifiers 33
4.3.1 Linguistic Variables 34
4.3.2 The Linguistic Variable Truth 35
Question Bank *38*
References *39*

Chapter 5: The Theory of Approximate Reasoning **41-53**
5.1 Introduction 41
5.2 Translation Rules 43
5.2.1 Entailment Rule 43
5.2.2 Conjunction Rule 43
5.2.3 Disjunction Rule 43
5.2.4 Projection Rule 44
5.2.5 Negation Rule 44
5.2.6 Compositional Rule of Inference 44
5.3 Rational Properties 45
5.3.1 Basic Property 45
5.3.2 Total Indeterminance 46
5.3.3 Subset 46
5.3.4 Superset 47
Question Bank. *50*
References. *51*

Chapter 6: Fuzzy Rule-Based Systems **54-70**
6.1 Introduction 54
6.2 Triangular Norm 54
6.3 Triangular Conorm 55
6.4 *t*-norm-based Intersection 57
6.5 *t*-conorm-Based Union 57
6.6 Averaging Operators 58
6.6.1 An Averaging Operator is a Function 58
6.6.2 Ordered Weighted Averaging 60
6.7 Measure of Dispersion or Entropy of an Owa Vector 63
6.8 Mamdani System 66
6.9 Larsen System 66
6.10 Defuzzification 67
Question Bank *68*
References *68*

Chapter 7: Fuzzy Reasoning Schemes **71-80**
7.1 Introduction 71
7.2 Fuzzy Rule-base System 71
7.3 Inference Mechanisms In Fuzzy Rule-base Systems 72
7.3.1 Mamdani inference Mechanism 73
7.3.2 Tsukamoto inference Mechanism 73
7.3.3 Sugeno Inference Mechanism 75
7.3.4 Larsen inference Mechanism 77

7.3.5 Simplified Fuzzy Reasoning 77
Question Bank *79*
References *79*

Chapter 8: Fuzzy Logic Controllers **81-93**
8.1 Introduction 81
8.2 Basic Feedback Control System 81
8.3 Fuzzy Logic Controller 82
8.3.1 Two-Input-Single-Output (TISO) Fuzzy Systems 82
8.3.2 Mamdani Type of Fuzzy Logic Control 82
8.3.3 Fuzzy Logic Control Systems 84
8.4 Defuzzification Methods 86
8.4.1 Center-of-Area/Gravity 87
8.4.2 First-of-Maxima 87
8.4.3 Middle-of-Maxima 87
8.4.4 Max-Criterion 88
8.4.5 Height defuzzification 88
8.5 Effectivity Of Fuzzy Logic Control Systems 89
Question Bank *91*
References *91*

Chapter 9: Fuzzy Logic-Applications **94-120**
9.1 Why use Fuzzy Logic? 94
9.2 Applications of Fuzzy Logic 95
9.3 When Not to use Fuzzy Logic? 96
9.4 Fuzzy Logic Model for Prevention of Road Accidents 96
9.4.1 Traffic Accidents and Traffic Safety 96
9.4.2 Fuzzy Logic Approach 97
9.4.3 Application 97
9.4.4 Membership Functions 98
9.4.5 Rule Base 99
9.4.6 Output 99
9.4.7 Conclusions 99
9.5 Fuzzy Logic Model to Control Room Temperature 100
9.5.1 The Mechanics of Fuzzy Logic 100
9.5.2 Fuzzification 101
9.5.3 Rule Application 102
9.5.4 Defuzzification 103
9.5.5 Conclusions 104
9.6 Fuzzy Logic Model for Grading of Apples 104
9.6.1 Apple Defects Used in the Study 105

9.6.2 Materials and Methods 105
9.6.3 Application of Fuzzy Logic 106
9.6.4 Fuzzy Rules 108
9.6.5 Determination of Membership Functions 109
9.6.6 Defuzzification 110
9.6.7 Results and Discussion 111
9.6.8 Conclusion 112
9.7 An Introductory Example: Fuzzy v/s Non-fuzzy 112
9.7.1 The Non-Fuzzy Approach 112
9.7.2 The Fuzzy Approach 116
9.7.3 Some Observations 117
Question Bank *118*
References *118*

Part II: Neural Networks

Chapter 10: Neural Networks Fundamentals **121-128**
10.1 Introduction 121
10.2 Biological Neural Network 121
10.3 A Framework for Distributed Representation 122
10.3.1 Processing Units 123
10.3.2 Connections between Units 123
10.3.3 Activation and Output Rules 124
10.4 Network Topologies 125
10.5 Training of Artificial Neural Networks 125
10.5.1 Paradigms of Learning 125
10.5.2 Modifying Patterns of Connectivity 126
10.6 Notation and Terminology 126
10.6.1 Notation 126
10.6.2 Terminology 127
Question Bank *128*
References *128*

Chapter 11: Perceptron and Adaline **129-138**
11.1 Introduction 129
11.2 Networks with Threshold Activation Functions 129
11.3 Perceptron Learning Rule and Convergence Theorem 131
11.3.1 Perceptron Learning Rule 131
11.3.2 Convergence Theorem 131
11.4 Adaptive Linear Element (Adalime) 133

11.5 The Delta Rule 134
11.6 Exclusive-or Problem 135
11.7 Multi-layer Perceptrons Can do Everything 137
Question Bank *138*
References *138*
Chapter 12: Back-Propagation **139-156**
12.1 Introduction 139
12.2 Multi - Layer Feed - Forward Networks 139
12.3 The Generalised Delta Rule 140
12.3.1 Understanding Back-Propagation 142
12.4 Working with Back-propagation 143
12.4.1 Weight Adjustments with Sigmoid Activation Function 143
12.4.2 Learning Rate and Momentum 144
12.4.3 Learning Per Pattern 144
12.5 Other Activation Functions 146
12.6 Deficiencies of Back-propagation 146
12.6.1 Network Paralysis 148
12.6.2 Local Minima 148
12.7 Advanced Algorithms 148
12.8 How Good are Multi-layer Feed-forward Networks? 151
12.8.1 The Effect of the Number of Learning Samples 152
12.8.2 The Effect of the Number of Hidden Units 153
12.9 Applications 153
Question Bank *155*
References *155*
Chapter 13: Recurrent Networks **157-189**
13.1 Introduction 157
13.2 The Generalised Delta - Rule In Recurrent Networks 157
13.2.1 The Jordan Network 158
13.2.2 The Elman Network 159
13.2.3 Back-Propagation in Fully Recurrent Networks 161
13.3 The Hopfield Network 161
13.3.1 Description 162
13.3.2 Hopfield Network as Associative Memory 163
13.3.3 Neurons with graded response 164
13.3.4 Hopfield networks for optimization problems 164
13.4 Boltzmann Machines 165
Question Bank *167*
References *167*

Chapter 14: Self-Organising Networks **169**
14.1 Introduction 169
14.2 Competitive Learning 170
14.2.1 Clustering 170
14.2.2 Vector Quantisation 174
14.2.3 Counter propagation 174
14.2.4 Learning Vector Quantisation 176
14.3 Kohonen Network 177
14.4 Principal Component Networks 179
14.4.1 Normalized Hebbian Rule 180
14.4.2 Principal Component Extractor 181
14.4.3 More eigenvectors 181
14.5 Adaptive Resonance Theory 182
14.5.1 Background: Adaptive Resonance Theory 182
14.5.2 ART1: The Simplified Neural Network Model 183
14.5.3 Operation 184
14.5.4 ART 1: The Original Model 185
14.5.5 Normalization of the Original Model 186
14.5.6 Contrast enhancement 187
Question Bank *188*
References *188*

Chapter 15: Reinforcement Learning **190-198**
15.1 Introduction 190
15.2 The Critic 190
15.3 The Controller Network 191
15.4 Barto's Approach: The ASE-ACE Combination 192
15.4.1 Associative Search 193
15.4.2 Adaptive Critic 194
15.4.3 The Cart-Pole System 194
15.5 Reinforcement Learning Versus Optimal Control 195
Question Bank *197*
References *197*

Chapter 16: Neural Networks Applications **199-215**
16.1 Introduction 199
16.2 Robot Control 200
16.2.1 Forward Kinematics 200
16.2.2 Inverse Kinematics 200
16.2.3 Dynamics 201
16.2.4 Trajectory generation 201

16.2.5 End-Effector Positioning 201
16.2.5a Involvement of neural networks 202
16.2.6 Camera-Robot Coordination in Function Approximation 202
16.2.6a Approach-1: Feed-forward Networks 203
16.2.6b Approach 2: Topology conserving maps 206
16.2.7 Robot Arm Dynamics 207
16.3 Detection of Tool Breakage in Milling Operations 210
16.3.1 Unsupervised Adaptive Resonance Theory (ART) Neural Networks 211
16.3.2 Results and Discussion 213
Question Bank *215*
References *215*

Part III: Hybrid Fuzzy Neural Networks

Chapter 17: Hybrid Fuzzy Neural Networks 217-232
17.1 Introduction 217
17.2 Hybrid Systems 217
17.2.1 Sequential Hybrid Systems 217
17.2.2 Auxiliary Hybrid Systems 218
17.2.3 Embedded Hybrid Systems 218
17.3 Fuzzy Logic in Learning Algorithms 219
17.4 Fuzzy Neurons 220
17.5 Neural Networks as Pre-processors or Post-processors 221
17.6 Neural Networks as Tuners of Fuzzy Logic Systems 222
17.7 Advantages and Drawbacks of Neurofuzzy Systems 223
17.8 Committee of Networks 223
17.9 Fnn Architecture Based On Back Propagation 224
17.9.1 Strong L-R Representation of Fuzzy Numbers 226
17.9.2 Simulation 228
17.10 Adaptive Neuro-fuzzy Inference System (ANFIS) 229
17.10.1 ANFIS Structure 231
Question Bank *232*
References *232*

Chapter 18: Hybrid Fuzzy Neural Networks Applications 233-252
18.1 Introduction 233
18.2 Tool Breakage Monitoring System for end Milling 233
18.2.1 Methodology: Force signals in the end milling cutting process 234
18.2.2 Neural Networks 235
18.2.3 Experimental Design and System Development Experimental Design 237

18.2.4 Neural Network-BP System Development 238
18.2.5 Findings and Conclusions 241
18.3 Control of Combustion 243
18.3.1 Adaptive neuro-fuzzy inference system 243
18.3.2 Learning Method of ANFIS 245
18.3.3 Model of Combustion 246
18.3.4 Optimization of the PI-Controllers using Genetic Algorithms 247
Question Bank *251*
References *251*
Index ***253***

This page intentionally left blank

CHAPTER 1

Introduction

Nowadays, *fuzzy logic*, *neural networks* have rooted in many application areas (expert systems, pattern recognition, system control, etc.). Although these methodologies seem to be different, they have many common features - like the use of basis functions (fuzzy logic has membership functions and neural networks have activation functions) and the aim to estimate functions from sample data or heuristics.

Fuzzy logic is mainly associated to imprecision, approximate reasoning and computing with words, and neural networks to learning and curve fitting (also to classification). These methods have in common that they

- are non-linear,
- have ability to deal with non-linearities,
- follow more human-like reasoning paths than classical methods,
- utilize self-learning,

1.1 FUZZY LOGIC (FL)

The concept of Fuzzy Logic was conceived by Lotfi A. Zadeh, a professor at the University of California at Berkley, and presented not as a control methodology, but as a way of processing data by allowing partial set membership rather than crisp set membership or non-membership. This approach to set theory was not applied to control systems until the 70's due to insufficient small-computer capability prior to that time. Professor Zadeh reasoned that people do not require precise, numerical information input, and yet they are capable of highly adaptive control. If feedback controllers could be programmed to accept noisy, imprecise input, they would be much more effective and perhaps easier to implement. Unfortunately, U.S. manufacturers have not been so quick to embrace this technology while the Europeans and Japanese have been aggressively building real products around it.

Basically, FL is a multivalued logic that allows intermediate values to be defined between conventional evaluations like true/false, yes/no, high/low, etc. Notions like rather tall or very fast can be formulated mathematically and processed by computers, in order to apply a more human-like way of thinking in the programming of computers.

Fuzzy systems is an alternative to traditional notions of set membership and logic that has its origins in ancient Greek philosophy. The precision of mathematics owes its success in large part to the efforts of Aristotle and the philosophers who preceded him. In their efforts to devise a concise theory of logic, and later mathematics, the so-called "Laws of Thought" were posited. One of these, the "Law of the Excluded Middle," states that every proposition must either be True or False. Even when Parminedes proposed the first version of this law (around 400 B.C.) there were strong and immediate objections: for example, Heraclitus proposed that things could be simultaneously True and not True. It was Plato who laid the foundation for what would become fuzzy logic, indicating that there was a third region (beyond True and False) where these opposites "tumbled about". But it was Lukasiewicz who first proposed a systematic alternative to the bi-valued logic of Aristotle.

If the conventional techniques of system analysis cannot be successfully incorporated to the modeling or control problem, the use of heuristic linguistic rules may be the most reasonable solution to the problem. For example, there is no mathematical model for truck and trailer reversing problem, in which the truck must be guided from an arbitrary initial position to a desired final position. Humans and fuzzy systems can perform this nonlinear control task with relative ease by using practical and at the same time imprecise rules as "If the trailer turns slightly left, then turn the wheel slightly left".

The most significant application area of FL has been in control field. It has been made a rough guess that 90% of applications are in control (the main part deals with rather simple applications, see Fig. 1.1). Fuzzy control includes fans, complex aircraft engines and control surfaces, helicopter control, missile guidance, automatic transmission, wheel slip control, industrial processes and so on. Commercially most significant have been various household and entertainment electronics, for example washing machine controllers and autofocus cameras. The most famous controller is the subway train controller in Sengai, Japan. Fuzzy system performs better (uses less fuel, drives smoother) when compared with a conventional PID controller. Companies that have fuzzy research are General Electric, Siemens, Nissan, Mitsubishi, Honda, Sharp, Hitachi, Canon, Samsung, Omron, Fuji, McDonnell Douglas, Rockwell, etc.

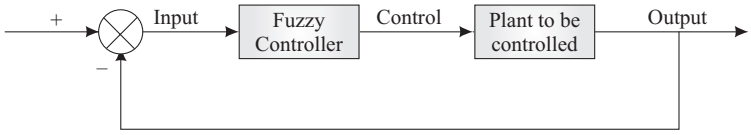

Fig. 1.1 Example of a control problem.

1.2 NEURAL NETWORKS (NN)

The study of neural networks started by the publication of Mc Culloch and Pitts [1943]. The single-layer networks, with threshold activation functions, were introduced by Rosenblatt [1959]. These types of networks were called perceptrons. In the 1960s it was experimentally shown that perceptrons could solve many problems, but many problems, which did not seem to be more difficult could not be solved. These limitations of one-layer perceptron were mathematically shown by Minsky and Papert in their book *Perceptron* [1969]. The result of this publication was that the neural networks lost their

interestingness for almost two decades. In the mid-1980s, back-propagation algorithm was reported by Rumelhart, Hinton, and Williams [1986], which revived the study of neural networks. The significance of this new algorithm was that multiplayer networks could be trained by using it.

NN makes an attempt to simulate human brain. The simulating is based on the present knowledge of brain function, and this knowledge is even at its best primitive. So, it is not absolutely wrong to claim that artificial neural networks probably have no close relationship to operation of human brains. The operation of brain is believed to be based on simple basic elements called neurons, which are connected to each other with transmission lines called axons and receptive lines called dendrites (see Fig. 1.2). The learning may be based on two mechanisms: the creation of new connections, and the modification of connections. Each neuron has an activation level which, in contrast to Boolean logic, ranges between some minimum and maximum value.

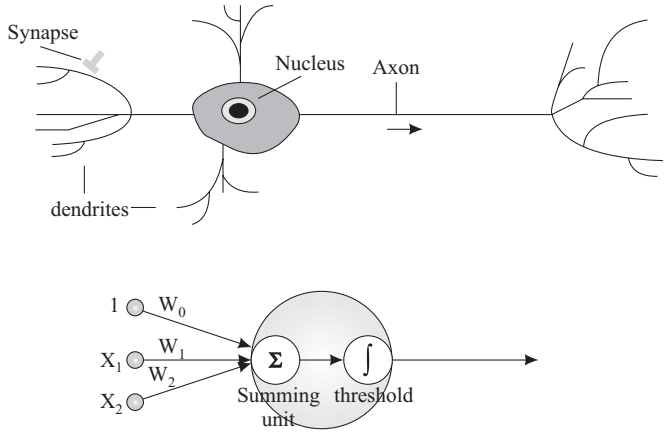

Fig. 1.2 Simple illustration of biological and artificial neuron (perceptron).

In artificial neural networks the inputs of the neuron are combined in a linear way with different weights. The result of this combination is then fed into a non-linear activation unit (activation function), which can in its simplest form be a threshold unit (See Fig. 1.2).

Neural networks are often used to enhance and optimize fuzzy logic based systems, e.g., by giving them a learning ability. This learning ability is achieved by presenting a training set of different examples to the network and using learning algorithm, which changes the weights (or the parameters of activation functions) in such a way that the network will reproduce a correct output with the correct input values. The difficulty is how to guarantee generalization and to determine when the network is sufficiently trained.

Neural networks offer nonlinearity, input-output mapping, adaptivity and fault tolerance. Nonlinearity is a desired property if the generator of input signal is inherently nonlinear. The high connectivity of the network ensures that the influence of errors in a few terms will be minor, which ideally gives a high fault tolerance.

1.3 SIMILARITIES AND DISSIMILARITIES BETWEEN FL AND NN

There are similarities between fuzzy logic and neural networks:

- estimate functions from sample data
- do not require mathematical model
- are dynamic systems
- can be expressed as a graph which is made up of nodes and edges
- convert numerical inputs to numerical outputs
- process inexact information inexactly
- have the same state space
- produce bounded signals
- a set of n neurons defines n-dimensional fuzzy sets
- learn some unknown probability function
- can act as associative memories
- can model any system provided the number of nodes is sufficient.

The main dissimilarity between fuzzy logic and neural network is that FL uses heuristic knowledge to form rules and tunes these rules using sample data, whereas NN forms "rules" based entirely on data.

1.4 APPLICATIONS

Applications can be found in signal processing, pattern recognition, quality assurance and industrial inspection, business forecasting, speech processing, credit rating, adaptive process control, robotics control, natural-language understanding, etc. Possible new application areas are programming languages, user-friendly application interfaces, automaticized programming, computer networks, database management, fault diagnostics and information security.

In many cases, good results have been achieved by combining both the methods. The number of this kind of hybrid systems is growing. A very interesting combination is the neuro-fuzzy architecture, in which the good properties of both methods are attempted to bring together. Most neuro-fuzzy systems are fuzzy rule based systems in which techniques of neural networks are used for rule induction and calibration. Fuzzy logic may also be employed to improve the performance of optimization methods used with neural networks.

QUESTION BANK

1. What is the ancient philosophy of fuzzy logic?
2. What are the various applications of fuzzy logic?
3. What is the historical evolution of neural networks?
4. What are the similarities and dissimilarities between fuzzy logic and neural networks?
5. What are the various applications of neural networks?

REFERENCES

1. W.S. McCulloch and W. Pitts, A logical calculus of the ideas immanent in nervous activity, *Bulletin of Mathematical Biophysics*, Vol. 5, pp. 115-133, 1943.
2. F. Rosenblatt, *Principles of Neurodynamics*, New York: Spartan Books, pp. 23-26, 1959.
3. L.A. Zadeh, Fuzzy Sets, *Information and Control*, Vol. 8, 338-353, 1965.
4. S. Korner, "Laws of thought", *Encyclopedia of Philosophy*, Vol. 4, pp. 414- 417, MacMillan, NY: 1967.
5. C. Lejewski, "Jan Lukasiewicz," *Encyclopedia of Philosophy*, Vol. 5, pp. 104-107, MacMillan, NY: 1967.
6. M. Minsky and S. Papert, *Perceptrons: An Introduction to Computational Geometry*, The MTT Press, 1969.
7. L.A. Zadeh, Making computers think like people, *IEEE*. *Spectrum*, pp. 26-32, Vol. 8, 1984.
8. D.E. Rumelhart, G.E. Hinton and R.J. Williams, Learning representations by backpropagating errors, *Nature*, Vol. 323, pp. 533-536, 1986.
9. U.S. Loses Focus on Fuzzy Logic, *Machine Design*, June 21, 1990.
10. Europe Gets into Fuzzy Logic, *Electronics Engineering Times*, Nov. 11, 1991.
11. Smith, T., Why the Japanese are going in for this fuzzy logic, *Business Week*, pp. 39, Feb. 20, 1993.
12. L.A. Zadeh, Soft computing and fuzzy logic. *IEEE Software*, Vol. (November), pp. 48-56. 1994.

CHAPTER 2

Fuzzy Sets and Fuzzy Logic

2. 1 INTRODUCTION

Fuzzy sets were introduced by L.A Zadeh in 1965 to represent/manipulate data and information possessing nonstatistical uncertainties. It was specifically designed to mathematically represent uncertainty and vagueness and to provide formalized tools for dealing with the imprecision intrinsic to many problems.

Fuzzy logic provides an inference morphology that enables approximate human reasoning capabilities to be applied to knowledge-based systems. The theory of fuzzy logic provides a mathematical strength to capture the uncertainties associated with human cognitive processes, such as thinking and reasoning. The conventional approaches to knowledge representation lack the means for representing the meaning of fuzzy concepts. As a consequence, the approaches based on first order logic and classical probability theory do not provide an appropriate conceptual framework for dealing with the representation of commonsense knowledge, since such knowledge is by its nature both lexically imprecise and noncategorical. The development of fuzzy logic was motivated in large measure by the need for a conceptual frame work which can address the issue of uncertainty and lexical imprecision.

2.2 WHAT IS FUZZY LOGIC?

Fuzzy logic is all about the relative importance of precision: How important is it to be exactly right when a rough answer will do? All books on fuzzy logic begin with a few good quotes on this very topic, and this is no exception. Here is what some clever people have said in the past:

Precision is not truth.

—Henri Matisse

Sometimes the more measurable drives out the most important.

—Rene Dubos

Vagueness is no more to be done away with in the world of logic than friction in mechanics.

—Charles Sanders Peirce

I believe that nothing is unconditionally true, and hence I am opposed to every statement of positive truth and every man who makes it.

—H. L. Mencken

So far as the laws of mathematics refer to reality, they are not certain. And so far as they are certain, they do not refer to reality.

—Albert Einstein

As complexity rises, *precise statements lose meaning and meaningful statements lose precision*.

—L. A Zadeh

Some pearls of folk wisdom also echo these thoughts:

Don't lose sight of the forest for the trees.

Don't be penny wise and pound foolish.

Fuzzy logic is a fascinating area of research because it does a good job of trading off between significance and precision - something that humans have been managing for a very long time (Fig. 2.1). Fuzzy logic sometimes appears exotic or intimidating to those unfamiliar with it, but once you become acquainted with it, it seems almost surprising that no one attempted it sooner. In this sense fuzzy logic is both old and new because, although the modern and methodical science of fuzzy logic is still young, the concepts of fuzzy logic reach right down to our bones.

Fuzzy logic is a convenient way to map an input space to an output space. This is the starting point for everything else, and the great emphasis here is on the word "convenient." What do I mean by mapping input space to output space? Here are a few examples: You tell me how good your service was at a restaurant, and I'll tell you what the tip should be. You tell me how hot you want the water, and I'll adjust the faucet valve to the right setting. You tell me how far away the subject of your photograph is, and I'll focus the lens for you. You tell me how fast the car is going and how hard the motor is working, and I'll shift the gears for you.

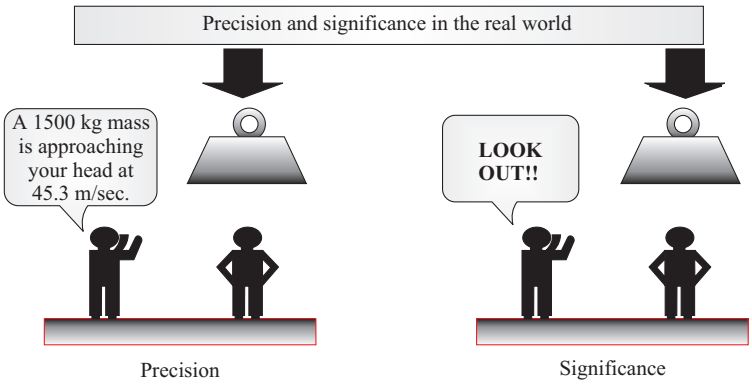
Precision and significance in the real world
A 1500 kg mass is approaching your head at 45.3 m/sec.
LOOK OUT!!
Precision
Significance

Fig. 2.1 Precision and significance.

2.3 HISTORICAL BACKGROUND

Almost forty years have passed since the publication of first paper on fuzzy sets. Where do we stand today? In viewing the evolution of fuzzy logic, three principal phases may be discerned.

The first phase, from 1965 to 1973, was concerned in the main with fuzzification, that is, with generalization of the concept of a set, with two-valued characteristic function generalized to a membership function taking values in the unit interval or, more generally, in a lattice. The basic issues and applications which were addressed were, for the most part, set-theoretic in nature, and logic and reasoning were not at the center of the stage.

The second phase, 1973-1999, two key concepts were introduced in this paper: (a) the concept of a linguistic variable; and (b) the concept of a fuzzy if-then rule. Today, almost all applications of fuzzy set theory and fuzzy logic involve the use of these concepts.

The term ***fuzzy logic*** was used for the first time in 1974. Today, ***fuzzy logic*** is used in two different senses: (a) a narrow sense, in which fuzzy logic, abbreviated as FLn, is a logical system which is a generalization of multivalued logic; and (b) a wide sense, in which fuzzy logic, abbreviated as FL, is a union of FLn, fuzzy set theory, possibility theory, calculus of fuzzy if-then rules, fuzzy arithmetic, calculus of fuzzy quantifiers and related concepts and calculi. The distinguishing characteristic of FL is that in FL everything is, or is allowed to be, a matter of degree. Today, the term ***fuzzy logic*** is used, for the most part, in its wide sense.

Perhaps the most striking development during the second phase of the evolution was the naissance and rapid growth of fuzzy control, alongside the boom in fuzzy logic applications, especially in Japan. There were many other major developments in fuzzy-logic-related basic and applied theories, among them the genesis of possibility theory and possibilistic logic, knowledge representation, decision analysis, cluster analysis, pattern recognition, fuzzy arithmetic; fuzzy mathematical programming, fuzzy topology and, more generally, fuzzy mathematics. Fuzzy control applications proliferated but their dominance in the literature became less pronounced.

Soft computing came into existence in 1981, with the launching of BISC (Berkeley Initiative in Soft Computing) at UC Berkeley. Basically, soft computing is a coalition of methodologies which collectively provide a foundation for conception, design and utilization of intelligent systems. The principal members of the coalition are: fuzzy logic, neurocomputing, evolutionary computing, probabilistic computing, chaotic computing, rough set theory and machine learning. The basic tenet of soft computing is that, in general, better results can be obtained through the use of constituent methodologies of soft computing in combination rather than in a stand-alone mode. A combination which has attained wide visibility and importance is that of neuro-fuzzy systems. Other combinations, e.g., neuro-fuzzy-genetic systems, are appearing, and the impact of soft computing is growing on both theoretical and applied levels.

An important development in the evolution of fuzzy logic, marking the beginning of the third phase, 1996 — is the genesis of computing with words and the computational theory of perceptions. Basically, development of computing with words and perceptions brings together earlier strands of fuzzy logic and suggests that scientific theories should be based on fuzzy logic rather than on Aristotelian, bivalent logic, as they are at present. A key component of computing with words is the concept of Precisiated Natural Language (PNL). PNL opens the door to a major enlargement of the role

of natural languages in scientific theories. It may well turn out to be the case that, in coming years, one of the most important application-areas of fuzzy logic, and especially PNL, will be the Internet, centering on the conception and design of search engines and question-answering systems.

From its inception, fuzzy logic has been (and to some degree still is) an object of skepticism and controversy. In part, skepticism about fuzzy logic is a reflection of the fact that, in English, the word ***fuzzy*** is usually used in a pejorative sense. But, more importantly, for some fuzzy logic is hard to accept because by abandoning bivalence it breaks with centuries-old tradition of basing scientific theories on bivalent logic. It may take some time for this to happen, but eventually abandonment of bivalence will be viewed as a logical development in the evolution of science and human thought.

2.4 CHARACTERISTICS OF FUZZY LOGIC

Some of the essential characteristics of fuzzy logic relate to the following:

- In fuzzy logic, exact reasoning is viewed as a limiting case of approximate reasoning.
- In fuzzy logic, everything is a matter of degree.
- In fuzzy logic, knowledge is interpreted a collection of elastic or, equivalently, fuzzy constraint on a collection of variables.
- Inference is viewed as a process of propagation of elastic constraints.
- Any logical system can be fuzzified.

2.5 CHARACTERISTICS OF FUZZY SYSTEMS

There are two main characteristics of fuzzy systems that give them better performance for specific applications:

- Fuzzy systems are suitable for uncertain or approximate reasoning, especially for the system with a mathematical model that is difficult to derive.
- Fuzzy logic allows decision making with estimated values under incomplete or uncertain information.

2.6 FUZZY SETS

2.6.1 Fuzzy Set

Let X be a nonempty set. A fuzzy set A in X is characterized by its membership function (Fig. 2.2).

$$\mu_A : X \rightarrow [0, 1] \qquad ...(2.1)$$

and $\mu_A(x)$ is interpreted as the degree of membership of element x in fuzzy set A for each $x \in X$.

It is clear that A is completely determined by the set of tuples

$$A = \{(u, \mu_A(u)) \mid u \in X\} \qquad ...(2.2)$$

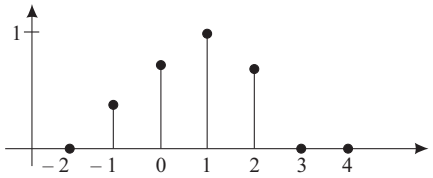

Fig. 2.2 A discrete membership function for "*x* is close to 1".

Frequently we will write $A(x)$ instead of $\mu A(x)$. The family of all fuzzy sets in X is denoted by $F(X)$. If $X = \{x_1, ..., x_n\}$ is a finite set and A is a fuzzy set in X then we often use the notation

$$A = \mu_1/x_1 + ... + \mu_n/x_n \quad ...(2.3)$$

where the term μ_i/x_i, $i = 1,...,n$ signifies that μ_i is the grade of membership of x_i in A and the plus sign represents the union.

Example 2.1: The membership function (Fig. 2.3) of the fuzzy set of real numbers "close to 1", can be defined as

$$A(t) = \exp(-\beta(t-1)^2)$$

where β is a positive real number.

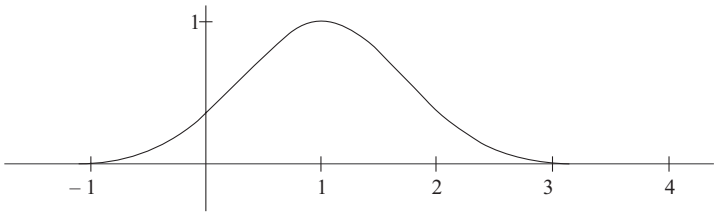

Fig. 2.3 A membership function for "*x* is close to 1".

Example 2.2: Assume someone wants to buy a cheap car. Cheap can be represented as a fuzzy set on a universe of prices, and depends on his purse (Fig. 2.4). For instance, from the Figure cheap is roughly interpreted as follows:

- Below Rs. 300000 cars are considered as cheap, and prices make no real difference to buyer's eyes.

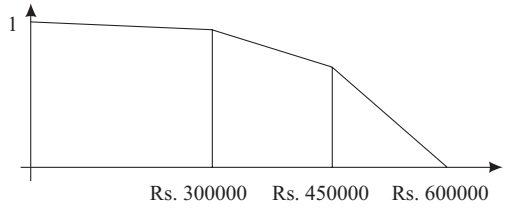

Fig. 2.4 Membership function of "cheap".

- Between Rs. 300000 and Rs. 450000, a variation in the price induces a weak preference in favor of the cheapest car.
- Between Rs. 450000and Rs. 600000, a small variation in the price induces a clear preference in favor of the cheapest car.
- Beyond Rs. 600000 the costs are too high (out of consideration).

2.6.2 Support

Let A be a fuzzy subset of X; the support of A, denoted supp (A), is the crisp subset of X whose elements all have nonzero membership grades in A.

$$\text{supp}(A) = \{x \in X \mid A(x) > 0\}. \qquad ...(2.5)$$

2.6.3 Normal Fuzzy Set

A fuzzy subset A of a classical set X is called normal if there exists an $x \in X$ such that $A(x) = 1$. Otherwise A is subnormal.

2.6.4 α-Cut

An a-level set of a fuzzy set A of X is a non-fuzzy set denoted by $[A]^{\alpha}$ and is defined by

$$[A]^{\alpha} = \begin{cases} \{t \varepsilon X | A(t) \geq \alpha\} & \text{if} \quad \alpha > 0 \\ cl(\text{supp } A) & \text{if} \quad \alpha = 0 \end{cases} \qquad ...(2.6)$$

where cl(supp A) denotes the closure of the support of A.

Example 2.3: Assume

$X = \{-2, -1, 0, 1, 2, 3, 4\}$ and

A = 0.0/–2 + 0.3/–1 + 0.6/0 + 1.0/1 + 0.6/2 + 0.3/3 + 0.0/4.

In this case

$$[A]^{\alpha} = \begin{cases} \{-1, 0, 1, 2, 3\} & \text{if} \quad 0 \leq \alpha \leq 0.3 \\ \{0, 1, 2\} & \text{if} \quad 0.3 < \alpha \leq 0.6 \\ \{1\} & \text{if} \quad 0.6 < \alpha \leq 1 \end{cases}$$

2.6.5 Convex Fuzzy Set

A fuzzy set A of X is called convex if $[A]^{\alpha}$ is a convex subset of $X \forall \alpha \in [0, 1]$. An α-cut of a triangular fuzzy number is shown in Fig. 2.5.

In many situations people are only able to characterize numeric information imprecisely. For example, people use terms such as, about 5000, near zero, or essentially bigger than 5000. These are examples of what are called fuzzy numbers. Using the theory of fuzzy subsets we can represent these fuzzy numbers as fuzzy subsets of the set of real numbers. More exactly,

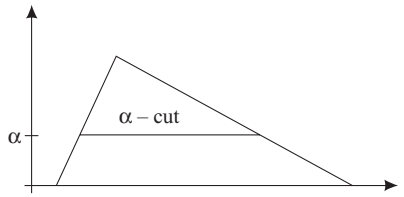

Fig. 2.5 An α-cut of a triangular fuzzy number.

2.6.6 Fuzzy Number

A fuzzy number (Fig. 2.6) A is a fuzzy set of the real line with a normal, (fuzzy) convex and continuous membership function of bounded support. The family of fuzzy numbers will be denoted by F .

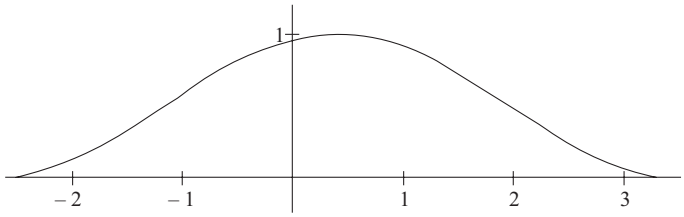

Fig. 2.6 Fuzzy number.

2.6.7 Quasi Fuzzy Number

A quasi-fuzzy number A is a fuzzy set of the real line with a normal, fuzzy convex and continuous membership function satisfying the limit conditions

$$\lim_{l \to \infty} A(t) = 0 \qquad ...(2.7)$$

Let Abe a fuzzy number. Then $[A]^{\gamma}$ is a closed convex (compact) subset of $\Re$ for all $\gamma \in [0,1]$. Let us introduce the notations

$$a_1(\gamma) = \min [A]^{\gamma}, \quad a_2(\gamma) \max [A]^{\gamma} \qquad ...(2.8)$$

In other words, a_1 (γ) denotes the left-hand side and a_2 (γ) denotes the right-hand side of the γ - cut. It is easy to see that

if $$\alpha \leq \beta \text{ then } [A]^{\alpha} \supset [A]^{\beta} \qquad ...(2.9)$$

Furthermore, the left-hand side function

$$a_1 : [0, 1] \to \Re \qquad ...(2.10)$$

is monotone increasing and lower semicontinuous, and the right-hand side function

$$a_2 : [0, 1] \to \Re \qquad ...(2.11)$$

is monotone decreasing and upper semicontinuous.

We shall use the notation

$$[A]^{\gamma} = [a_1(\gamma),\ a_2(\gamma)] \qquad ...(2.12)$$

The support of A is the open interval $[a_1(0), a_2(0)]$ and it is illustrated in Fig. 2.7.

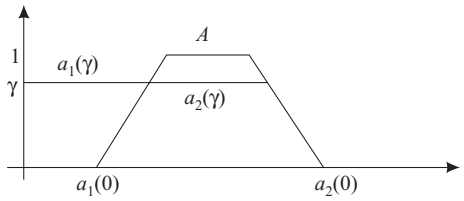

Fig. 2.7 The support of A is $[a_1(0), a_2(0)]$.

If A is not a fuzzy number then there exists an $\gamma \in [0, 1]$ such that $[A]^{\gamma}$ is not a convex subset of R. The not fuzzy number is shown in Fig. 2.8.

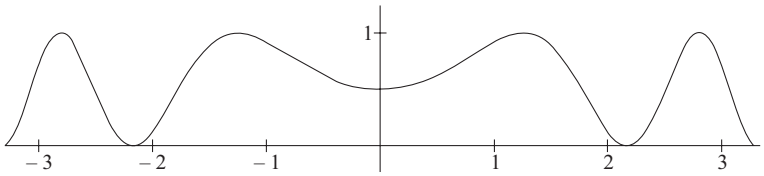

Fig. 2.8 Not fuzzy number.

2.6.8 Triangular Fuzzy Number

A fuzzy set A is called triangular fuzzy number with peak (or center) a, left width $\alpha > 0$ and right width $\beta > 0$ if its membership function has the following form

$$A(t) = \begin{cases} 1 - \dfrac{a - t}{\alpha} & \text{if } a - \alpha \le t \le a \\ 1 - \dfrac{a - t}{\beta} & \text{if } a \le t \le a + \beta \\ 0 & \text{otherwise} \end{cases} \qquad ...(2.13)$$

and we use the notation $A = (\alpha, \alpha, \beta)$. It can easily be verified that

$$[A]^{\gamma} = [a - (1 - \gamma)\alpha,\ a + (1 - \gamma)\beta],\ \forall_{\gamma} \in [0,1] \qquad ...(2.14)$$

The support of A is $(a - \alpha, b + \beta)$.

A triangular fuzzy number (Fig. 2.9) with center a may be seen as a fuzzy quantity "x is approximately equal to a".

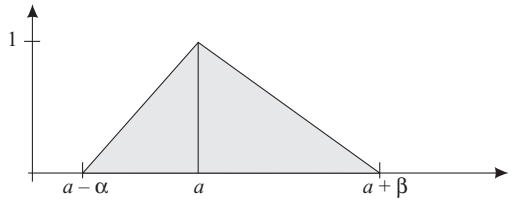

Fig. 2.9 Triangular fuzzy number.

2.6.9 Trapezoidal Fuzzy Number

A fuzzy set A is called trapezoidal fuzzy number with tolerance interval $[a, b]$, left width and right width β if its membership function has the following form.

$$A(t) = \begin{cases} 1 - \dfrac{a - t}{\alpha} & \text{if } a - \alpha \le t \le a \\ 1 & \text{if } a \le t \le b \\ 1 - \dfrac{t - b}{\beta} & \text{if } a \le t \le b + \beta \\ 0 & \text{otherwise} \end{cases} \quad ...(2.15)$$

and we use the notation $A = (a, b, \alpha, \beta)$. It can easily be shown that

$$[A]^{\gamma} = [a - (1 - \gamma)\alpha,\ b + (1 - \gamma)\beta],\ \forall_{\gamma} \in [0, 1] \quad ...(2.16)$$

The support of is $(a - \alpha, b + \beta)$.

A trapezoidal fuzzy number (Fig. 2.10) may be seen as a fuzzy quantity "x is approximately in the interval $[a, b]$".

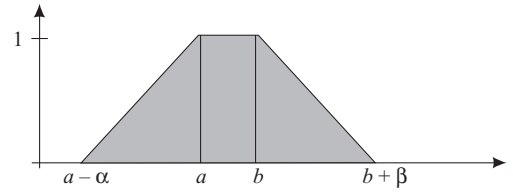

Fig. 2.10 Trapezoidal fuzzy number.

2.6.10 Subsethood

Let A and B are fuzzy subsets of a classical set X. We say that A is a subset of B if $A(t) \le B(t)$, $\forall t \in X$. The subsethood is illustrated in Fig. 2.11.

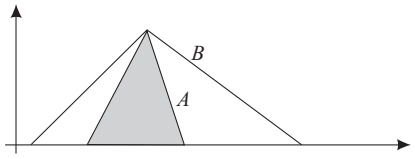

Fig. 2.11 *A* is a subset of *B*.

2.6.11 Equality of Fuzzy Sets

Let A and B are fuzzy subsets of a classical set X. A and B are said to be equal, denoted $A = B$, if $A \subset B$ and $B \subset A$. We note that $A = B$ if and only if $A(x) = B(x)$ for $x \in X$.

2.6.12 Empty Fuzzy Set

The empty fuzzy subset of X is defined as the fuzzy subset Ø of X such that $Ø(x) = 0$ for each $x \in X$. It is easy to see that $Ø \subset A$ holds for any fuzzy subset A of X.

2.6.13 Universal Fuzzy Set

The largest fuzzy set in X, called universal fuzzy set (Fig. 2.12) in X, denoted by 1_X, is defined by $1_X(t) = 1$, $\forall t \in X$. It is easy to see that $A \subset 1_X$ holds for any fuzzy subset A of X.

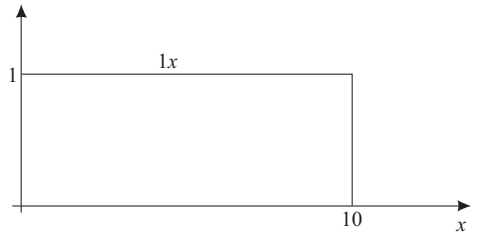

Fig. 2.12 The graph of the universal fuzzy subset in $X = [0, 10]$.

2.6.14 Fuzzy Point

Let A be a fuzzy number. If supp $(A) = \{x_0\}$, then A is called a fuzzy point (Fig. 2.13) and we use the notation $A = \bar{x}_0$.

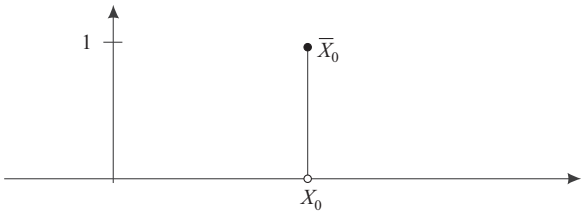

Fig. 2.13 Fuzzy point.

Let $A = \bar{x}_0$ be a fuzzy point. It is easy to see that

$$[A]^{\gamma} = [x_0, x_0] = \{x_0\}, \forall_{\gamma} \in [0, 1] \quad ...(2.17)$$

2.7 OPERATIONS ON FUZZY SETS

We extend the classical set theoretic operations from ordinary set theory to fuzzy sets. We note that all those operations which are extensions of crisp concepts reduce to their usual meaning when the fuzzy subsets have membership degrees that are drawn from {0, 1}. For this reason, when extending operations to fuzzy sets we use the same symbol as in set theory.

Let A and B are fuzzy subsets of a nonempty (crisp) set X.

2.7.1 Intersection

The intersection of A and B is defined as

$$(A \cap B)(t) = \min \{A(t), B(t)\} = A(t) \wedge B(t) \text{ for all } t \in X \quad ...(2.18)$$

The intersection of A and B is shown in Fig. 2.14.

2.7.2 Union

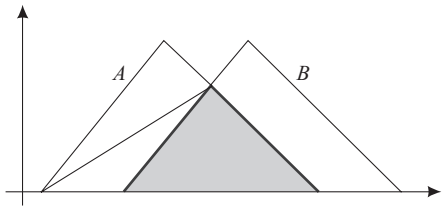

Fig. 2.14 Intersection of two triangular fuzzy numbers.

The union of A and B is defined as

$$(A \cup B)(t) = \max \{A(t), \quad B(t)\} = A(t) \vee B(t) \text{ for all } t \in X \quad ...(2.19)$$

The union of two triangular numbers is shown in Fig. 2.15.

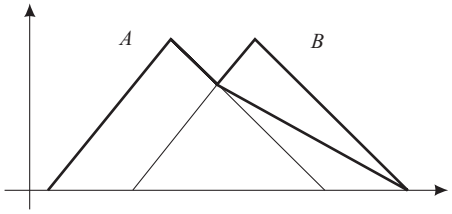

Fig. 2.15 Union of two triangular fuzzy numbers.

2.7.3 Complement

The complement of a fuzzy set A is defined as

$$(\neg A)(t) = 1 - A(t) \quad ...(2.20)$$

A closely related pair of properties which hold in ordinary set theory are the law of excluded middle

$$A \vee \neg A = X \quad ...(2.21)$$

and the law of non-contradiction principle

$$A \wedge \neg A = \phi \quad ...(2.22)$$

It is clear that $\neg 1_X = \phi$ and $\neg \phi = 1_X$, however, the laws of excluded middle and noncontradiction are not satisfied in fuzzy logic.

Lemma–2.1: The law of excluded middle is not valid. Let $A(t) = 1/2$, $\forall t \in R$, then it is easy to see that

$$\begin{aligned}(\neg A \vee A)(t) &= \max \{\neg A(t), A(t)\} \\ &= \max \{1 - 1/2, 1/2\} \\ &= 1/2 \neq 1\end{aligned}$$

Lemma–2.2: The law of non-contradiction is not valid. Let $A(t) = 1/2$, $\forall t \in R$, then it is easy to see that

$$\begin{aligned}(\neg A \vee A)(t) &= \text{mix} \{\neg A(t), A(t)\} \\ &= \text{mix} \{1 - 1/2, 1/2\} \\ &= 1/2 \neq 0\end{aligned}$$

However, fuzzy logic does satisfy De Morgan's laws

$$\neg(A \wedge B) = \neg A \vee \neg B, \quad \neg(A \vee B = \neg A \wedge \neg B)$$

QUESTION BANK

1. What is fuzzy logic?
2. Explain the evolution phases of fuzzy logic.
3. What are the characteristics of fuzzy logic?
4. What are the characteristics of fuzzy systems?
5. What are the different fuzzy sets? Define them.
6. What are the roles of α-cut in fuzzy set theory?
7. What are the different fuzzy numbers? Define them.
8. Define the following: (i) equality of fuzzy sets, (ii) empty fuzzy set, (iii) universal fuzzy set.
9. What are the operations on fuzzy sets? Explain with examples.
10. Given $A = \{a, b, c, 1, 2\}$ and $B = \{1, 2, 3, b, c\}$. Find $A \cup B$, and $A \cap B$.
11. Given $X = \{1, 2, 3, 4, 5, 6,\}$ and $A = \{2, 4, 6\}$. Find $\neg A$.
12. Let A be a fuzzy set defined by
    $A = 0.5/x_1 + 0.4/x_2 + 0.7/x_3 + 0.8/x_4 + 1/x_5$. List all α-cuts.

REFERENCES

1. L.A. Zadeh, Fuzzy Sets, *Information and Control*, Vol. 8, 338-353, 1965.
2. L.A. Zadeh, Fuzzy algorithms, *Information and Control*, Vol. 12, pp. 94-102, 1968.
3. J. G. Brown, *A* note on fuzzy sets, *Information and Control*, Vol. 18, No. 1, pp. 32-39, 1971.
4. L.A. Zadeh, *A* fuzzy-set-theoretic interpretation of linguistic hedges. *Journal of Cybernetics*, Vol. 2, No. 2, pp. 4-34. 1972.
5. A. DeLuca, and S. Termini, Algebraic properties of fuzzy sets, *Journal of Mathematics Analysis and Applications*, Vol. 40, No. 2, pp. 373-386, 1972.
6. L.A. Zadeh, Outline of a new approach to the analysis of complex systems and decision process, *IEEE* Transactions on systems, *Man and Cybernetics*, Vol. *SMC*-3, No. 1, pp. 28-44, 1973.
7. L.A. Zadeh, The concept of a linguistic variable and its application to approximate reasoning, Part 1, *Information Sciences*, Vol. 8, pp. 199-249. 1974.
8. L.A. Zadeh, The concept of a linguistic variable and its application to approximate reasoning, Part 2, *Information Sciences*, Vol. 8, pp. 301-357. 1975.
9. P. Albert, The algebra of fuzzy logic, *Fuzzy Sets and Systems*, Vol. 1, No. 3, pp. 203-230, 1978.
10. Dubois, D. and H. Prade, Operations on Fuzzy Numbers, *International Journal of Systems Science*, Vol. 9, pp. 613-626, 1978.
11. S. Watanable, A generalized fuzzy set theory, *IEEE Transactions*, on systems, man and cybernetics, Vol. 8, No. 10, pp. 756-759, 1978.
12. S. Gottwald, Set theory for fuzzy sets of higher level, *Fuzzy Sets and Systems*, Vol. 2, No. 2, pp. 125-151, 1979.
13. L.A. Zadeh, Possibility theory and soft data analysis, Mathematics Frontiers of the Social and Policy Sciences, L.Cobb and R.M. Throll (eds.), pp. 69-129, Westview Press, Boulder, 1981.
14. L.A. Zadeh, Making computers think like people, *IEEE Spectrum*, Vol.8, pp. 26-32, 1984.
15. L.A. Zadeh, Fuzzy logic, *IEEE Computer*, Vol. 21, No. 4, pp. 83-93, 1988.
16. U. Hohle and L.N. Stout, Foundations of fuzzy sets, *Fuzzy Sets and Systems*, Vol. 40, No. 2, pp.257-296, 1991.
17. L.A. Zadeh, Soft computing and fuzzy logic. *IEEE Software*, Vol. (November), pp. 48-56. 1994.
18. L.A. Zadeh, Fuzzy logic, neutral networks and soft computing, *Communications of the ACM*, Vol. 37, No. 3, pp. 77-84, 1994.
19. L.A. Zadeh, Fuzzy logic–computing with words, *IEEE Transactions on Fuzzy Systems*, Vol.4, No. 2, pp. 103-111, 1996.
20. L.A. Zadeh, Roles of soft computing and fuzzy logic in the conception, design and deployment of information/intelligent systems, Computational Intelligence: Soft Computing and Fuzzy-Neuro Integration with Applications, edited by O. Kaynak, L. A. Zadeh, B. Turksen, and I. J. Rudas, Springer-Verlag, Berlin, pp.10-37, 1998.

CHAPTER 3

Fuzzy Relations

3.1 INTRODUCTION

A classical relation can be considered as a set of tuples, where a tuple is an ordered pair. A binary tuple is denoted by (u, v), an example of a ternary tuple is (u, v, w) and an example of n-ary tuple is $(X_1, ..., X_n)$.

Example 3.1: Let X be the domain of man {John, Charles, James} and Y the domain of women {Diana, Rita, Eva}, then the relation "married to" on $X \times Y$ is, for example

{(Charles, Diana), (John, Eva), (James, Rita)}

3.2 FUZZY RELATIONS

3.2.1 Classical N-Array Relation

Let $X_1, ..., X_n$ be classical sets. The subsets of the Cartesian product $X_1 \; x \; \ldots x \; X_n$ are called n-ary relations. If $X_1 = ... = X_n$ and $R \subset X^n$, then R is called an n-ary relation in X. Let R be a binary relation in $\Re$. Then the characteristic function of R is defined as

$$X_R(u, v) = \begin{cases} 1 & \text{if} \quad (u, v) \in R \\ 0 & \text{otherwise} \end{cases} \qquad ...(3.1)$$

Example 3.2 Consider the following relation

$$(u, v) \in R \Leftrightarrow u \in [a, b] \text{ and } \quad v \in [0, c] \qquad ...(3.2)$$

$$X_R(u, v) = \begin{cases} 1 & \text{if} \quad (u, v) \in [a,b] x [0, c] \\ 0 & \text{otherwise} \end{cases}$$

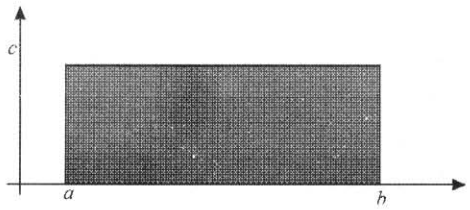

Fig. 3.1

3.2.2 Reflexivity

Let R be a binary relation in a classical set X. Then R is reflexive if $\forall u \in U$: $(u, v) \in R$.

3.2.3 Anti-Reflexivity

R is anti-reflexive if $\forall u \in U$: $(u, v) \notin R$.

3.2.4 Symmetricity

R is symmetric if from $(u, v) \in R \rightarrow (u, v) \in R$, $\forall u, v \in U$.

3.2.5 Anti-Symmetricity

R is anti-symmetric if $(u, v) \in R$ and $(v, u) \in R$ then $u = v$, $\forall u, v \in U$.

3.2.6 Transitivity

R is transitive if $(u, v) \in R$ and $(v, w) \in R$ then $(u, w) \in R$, $\forall u, v, w \in U$.

Example 3.3: Consider the classical inequality relations on the real line $\Re$. It is clear that $\leq$ is reflexive, anti-symmetric and transitive, $<$ is anti-reflexive, anti-symmetric and transitive.

Other important properties of binary relations are:

3.2.7 Equivalence

R is an equivalence relation if, R is reflexive, symmetric and transitive.

3.2.8 Partial Order

R is a partial order relation if it is reflexive, anti-symmetric and transitive.

3.2.9 Total Order

R is a total order relation if it is partial order and $\forall u, v \in R$, $(u, v) \in R$ or $(v, u) \in R$ hold.

Example 3.4: Let us consider the binary relation "subset of". It is clear that we have a partial order relation. The relation $\leq$ on natural numbers is a total order relation.

Consider the relation "mod 3" on natural numbers

$$\{(m, n) \mid (n - m) \bmod 3 \equiv 0\}$$

This is an equivalence relation.

3.2.10 Binary Fuzzy Relation

Let X and Y be nonempty sets. A fuzzy relation R is a fuzzy subset of $X \times Y$.
In other words, $R \in F(X \times Y)$.

If $X = Y$ then we say that R is a binary fuzzy relation in X.

Let R be a binary fuzzy relation on R. Then $R(u, v)$ is interpreted as the degree of membership of the ordered pair (u, v) in R.

Example 3.5: A simple example of a binary fuzzy relation on $U = \{1, 2, 3\}$, called "approximately equal" can be defined as

$$R(1, 1) = R(2, 2) = R(3, 3) = 1$$

$$R(1, 2) = R(2, 1) = R(2, 3) = R(3, 2) = 0.8$$

$$R(1, 3) = R(3, 1) = 0.3$$

The membership function of R is given by

$$R(u, v) = \begin{cases} 1 & \text{if} \quad u = v \\ 0.8 & \text{if} \quad |u - v| = 1 \\ 0.3 & \text{if} \quad |u - v| = 2 \end{cases}$$

In matrix notation it can be represented as

$$\begin{bmatrix} & 1 & 2 & 3 \\ 1 & 1 & 0.8 & 0.3 \\ 2 & 0.8 & 1 & 0.8 \\ 3 & 0.3 & 0.8 & 1 \end{bmatrix}$$

3.3 OPERATIONS ON FUZZY RELATIONS

Fuzzy relations are very important because they can describe interactions between variables. Let R and S be two binary fuzzy relations on $X \times Y$.

3.3.1 Intersection

The intersection of R and S is defined by

$$(R \wedge S)(u, v) = \min \{R(u, v), S(u, v)\} \qquad ...(3.3)$$

Note that R: $X \times Y \to [0, 1]$, i.e. R the domain of R is the whole Cartesian product $X \times Y$.

3.3.2 Union

The union of R and S is defined by

$$(R \vee S)(u, v) = \min\{R(u, v), S(u, v)\} \quad ...(3.4)$$

Example 3.6: Let us define two binary relations

R = "x is considerable larger than y"

$$\begin{bmatrix} & y_1 & y_2 & y_3 & y_4 \\ x_1 & 0.8 & 0.1 & 0.1 & 0.7 \\ x_2 & 0 & 0.8 & 0 & 0 \\ x_3 & 0.9 & 1 & 0.7 & 0.8 \end{bmatrix}$$

S = "x is very close to y"

$$\begin{bmatrix} & y_1 & y_2 & y_3 & y_4 \\ x_1 & 0.4 & 0 & 0.9 & 0.6 \\ x_2 & 0.9 & 0.4 & 0.5 & 0.7 \\ x_3 & 0.3 & 0 & 0.8 & 0.5 \end{bmatrix}$$

The intersection of R and S means that "x is considerable larger than y" and "x is very close to y".

$$(R \wedge S)(x, y) = \begin{bmatrix} & y_1 & y_2 & y_3 & y_4 \\ x_1 & 0.4 & 0 & 0.1 & 0.6 \\ x_2 & 0 & 0.4 & 0 & 0 \\ x_3 & 0.3 & 0 & 0.7 & 0.5 \end{bmatrix}$$

The union of R and S means that "x is considerable larger than y" or "x is very close to y".

$$(R \vee S)(x, y) = \begin{bmatrix} & y_1 & y_2 & y_3 & y_4 \\ x_1 & 0.8 & 0 & 0.9 & 0.7 \\ x_2 & 0.9 & 0.8 & 0.5 & 0.7 \\ x_3 & 0.9 & 1 & 0.8 & 0.8 \end{bmatrix}$$

Consider a classical relation R on $\Re$.

$$R(u, v) = \begin{cases} 1 & \text{if} \quad (u, v) \in [a, b] x [0, c] \\ 0 & \text{otherwise} \end{cases} \quad ...(3.5)$$

It is clear that the projection (or shadow) of R on the X-axis is the closed interval $[a, b]$ and its projection on the Y-axis is $[0, c]$.

If R is a classical relation in $X \times Y$, then

$$\prod_x = \{x \in X | \exists y \in Y (x, y) \in R\} \quad ...(3.6)$$

$$\prod_y = \{y \in Y | \exists x \in X (x, y) \in R\} \quad ...(3.7)$$

where $\prod_x$ denotes projection on X and $\prod_y$ denotes projection on Y.

3.3.3 Projection

Let R be a fuzzy binary fuzzy relation on $X \times Y$. The projection of R on X is defined as

$$\prod_x (x) = \sup \{R(x, y) \mid y \in Y\} \quad ...(3.8)$$

and the projection of R on Y is defined as

$$\prod_y (y) = \sup \{R(x, y) \mid x \in X\} \quad ...(3.9)$$

Example 3.7: Consider the relation

$$R = \text{“}x \text{ is considerable larger than } y\text{”} = \begin{bmatrix} & y_1 & y_2 & y_3 & y_4 \\ x_1 & 0.8 & 0.1 & 0.1 & 0.7 \\ x_2 & 0 & 0.8 & 0 & 0 \\ x_3 & 0.9 & 1 & 0.7 & 0 \end{bmatrix}$$

then the projection on X means that

- x_1 is assigned the highest membership degree from the tuples (x_1, y_1), (x_1, y_2), (x_1, y_3), (x_1, y_4), i.e. $\prod_x(x_1) = 1$, which is the maximum of the first row.
- x_2 is assigned the highest membership degree from the tuples (x_2, y_1), (x_2, y_2), (x_2, y_3), (x_2, y_4), i.e. $\prod_x(x_2) = 0.8$, which is the maximum of the second row.
- x_3 is assigned the highest membership degree from the tuples (x_3, y_1), (x_3, y_2), (x_3, y_3), (x_3, y_4), i.e. $\prod_x(x_3) = 1$, which is the maximum of the third row.

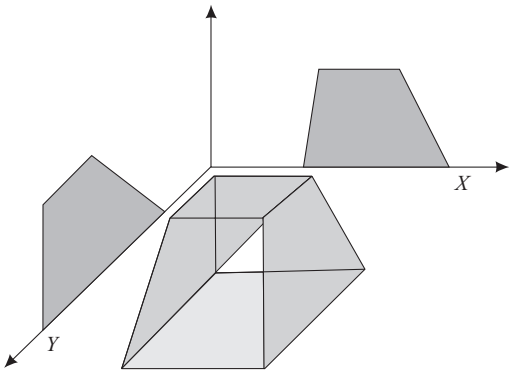

Fig. 3.2 Shadows of a fuzzy relation.

3.3.4 Cartesian Product of Two Fuzzy Sets

The Cartesian product of $A \in F(X)$ and $B \in F(Y)$ is defined as

$$(A \times B)(u, v) = \min\{A(u), B(v)\} \quad ...(3.10)$$

for all $u \in X$ and $v \in Y$.

It is clear that the Cartesian product of two fuzzy sets (Fig. 3.3) is a fuzzy relation in $X \times Y$.

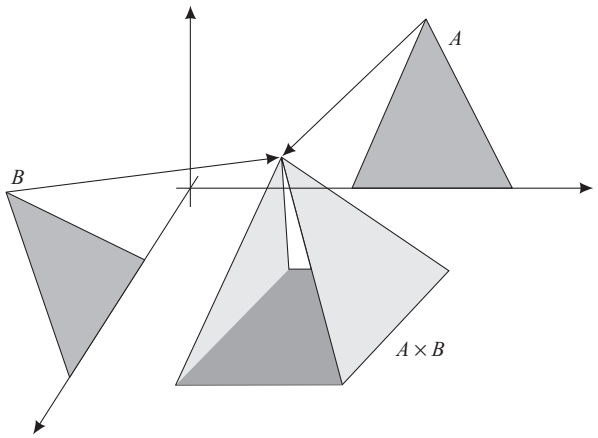

Fig. 3.3 Cartesian product of two fuzzy sets.

If A and B are normal, then $\prod_y (A \times B) = B$ and $\prod_x (A \times B) = A$.

Really,

$$\begin{aligned} \prod_x(x) &= \sup\{A \times B(x, y) | y\} \\ &= \sup\{A(x) \wedge B(y) | y\} \\ &= \min\{A(x), \sup\{B(y)\} | y\} \\ &= \min\{A(x), 1\} \\ &= A(x) \end{aligned} \quad ...(3.11)$$

3.3.5 Shadow of Fuzzy Relation

The sup-min composition of a fuzzy set $C \in F(X)$ and a fuzzy relation $R \in F(X \times Y)$ is defined as

$$(C \circ R)(y) = \sup \min\{C(x), R(x, y)\} \quad ...(3.12)$$

for all $x \in X$ and $y \in Y$.

The composition of a fuzzy set C and a fuzzy relation R can be considered as the shadow of the relation R on the fuzzy set C (Fig. 3.4).

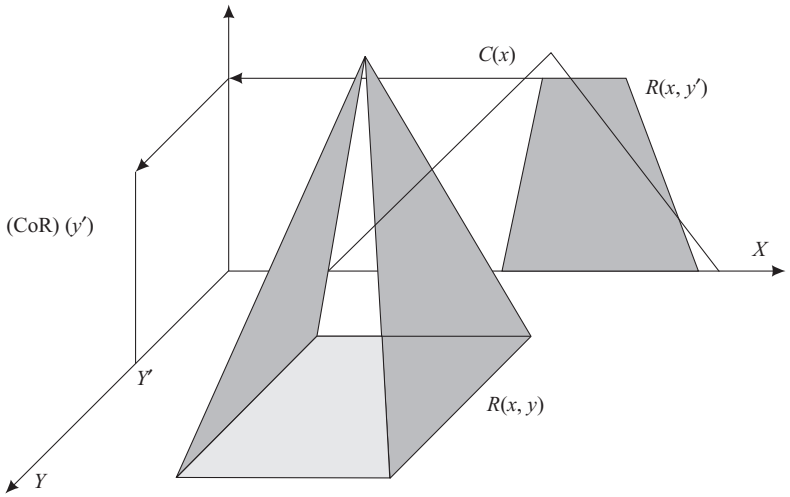

Fig. 3.4 Shadow of fuzzy relation *R* on the fuzzy set *C*.

Example 3.8: Let *A* and *B* be fuzzy numbers and let

$$R = A \times B$$

a fuzzy relation.

Observe the following property of composition

$$A \circ R = A \circ (A \times B) = A$$

$$B \circ R = B \circ (A \times B) = B$$

Example 3.9: Let *C* be a fuzzy set in the universe of discourse {1, 2, 3} and let *R* be a binary fuzzy relation in {1, 2, 3}. Assume that $C = 0.2/1 + 1/2 + 0.2/3$ and

$$R = \begin{bmatrix} & 1 & 2 & 3 \\ 1 & 1 & 0.8 & 0.3 \\ 2 & 0.8 & 1 & 0.8 \\ 3 & 0.3 & 0.8 & 1 \end{bmatrix}$$

Using the definition of sup-min composition we get

$$C \circ R = (0.2/1 + 1/2 + 0.2/3) \circ \begin{bmatrix} & 1 & 2 & 3 \\ 1 & 1 & 0.8 & 0.3 \\ 2 & 0.8 & 1 & 0.8 \\ 3 & 0.3 & 0.8 & 1 \end{bmatrix} = 0.8/1 + ½ + 0.8/3$$

Example 3.10: Let C be a fuzzy set in the universe of discourse [0, 1] and let R be a binary fuzzy relation in [0, 1]. Assume that $C(x) = x$ and $R(x, y) = 1 - |x - y|$.

Using the definition of sup-min composition we get

$$C \text{ o } R(y) = \sup \min\{x, 1 - |x - y|\} = \frac{1 + y}{2}$$

for all $x \in [0, 1]$ and $y \in [0, 1]$

3.3.6 Sup-Min Composition of Fuzzy Relations

Let $R \in F(X \times Y)$ and $S \in F(Y \times Z)$. The sup-min composition of R and S, denoted by R o S is defined as

$$(R \text{ o } S)(u, w) = \sup \min\{R(u, v), S(v, w)\} \qquad ...(3.13)$$

for $v \in Y$

It is clear that R o S is a binary fuzzy relation in $X \times Z$.

Example 3.11: Consider two fuzzy relations

$$R = \text{“}x \text{ is considerable larger than } y\text{”} = \begin{bmatrix} & y_1 & y_2 & y_3 & y_4 \\ x_1 & 0.8 & 0.1 & 0.1 & 0.7 \\ x_2 & 0 & 0.8 & 0 & 0 \\ x_3 & 0.9 & 1 & 0.7 & 0.8 \end{bmatrix}$$

$$S = \text{“}y \text{ is very close to } z\text{”} = \begin{bmatrix} & z_1 & z_2 & z_3 \\ y_1 & 0.4 & 0.9 & 0.3 \\ y_2 & 0 & 0.4 & 0 \\ y_3 & 0.9 & 0.5 & 0.8 \\ y_4 & 0.6 & 0.7 & 0.5 \end{bmatrix}$$

Then their composition is

$$\text{R o S} = \begin{bmatrix} & z_1 & z_2 & z_3 \\ x_1 & 0.6 & 0.8 & 0.5 \\ x_2 & 0 & 0.4 & 0 \\ x_3 & 0.7 & 0.9 & 0.7 \end{bmatrix}$$

formally,

$$\begin{bmatrix} & y_1 & y_2 & y_3 & y_4 \\ x_1 & 0.8 & 0.1 & 0.1 & 0.7 \\ x_2 & 0 & 0.8 & 0 & 0 \\ x_3 & 0.9 & 1 & 0.7 & 0.8 \end{bmatrix} \text{ o } \begin{bmatrix} & z_1 & z_2 & z_3 \\ y_1 & 0.4 & 0.9 & 0.3 \\ y_2 & 0 & 0.4 & 0 \\ y_3 & 0.9 & 0.5 & 0.8 \\ y_4 & 0.6 & 0.7 & 0.5 \end{bmatrix} = \begin{bmatrix} & z_1 & z_2 & z_3 \\ x_1 & 0.6 & 0.8 & 0.5 \\ x_2 & 0 & 0.4 & 0 \\ x_3 & 0.7 & 0.9 & 0.7 \end{bmatrix}$$

i.e., the composition of R and S is nothing else, but the classical product of the matrices R and S with the difference that instead of addition we use maximum and instead of multiplication we use minimum operator.

QUESTION BANK

1. What are the fuzzy relations? Explain them.
2. Explain the operations on the fuzzy relations.
3. Given any N-ary relation, how many different projections of the relation can be taken?
4. Given $A = \{(x_1, 0.1), (x_2, 0.5), (x_3, 0.3)\}$ and $B = \{(y_1, 0.3), (y_2, 0.4)\}$ be the two fuzzy sets on the universes of discourse $X = \{x_1, x_2, x_3\}$ and $Y = \{y_1, y_2\}$ respectively. Find the Cartesian product of A and B.
5. Given $X = \{x_1, x_2, x_3, x_4\}$ of four varieties of paddy plants, $D = \{d_1, d_2, d_3, d_4\}$ of the various diseases affecting the plants and $Y = \{y_1, y_2, y_3, y_4\}$ be the common symptoms of the diseases. Find SUP-MIN composition.

REFERENCES

1. L.A. Zadeh, Fuzzy Sets, *Information and Control*, Vol. 8, 338-353, 1965.
2. L.A, Zadeh, Similarity relations and fuzzy orderings, *Information Sciences*, Vol. 2, No. 2, pp. 177-200, 1971.
3. Dubois, D. and H. Prade, *Fuzzy Sets and Systems*: *Theory and Applications*, Academic Press, NY, 1980.
4. J.F. Bladwin, and N.C.F. Guild, Modelling Controllers Using Fuzzy Relations, *Kybernetes*, Vol. 9, No. 3, pp. 223-229, 1980.
5. R.R. Yager, Some properties of fuzzy relationships, *Cybernetics and Systems*, Vol. 12, No. 2, pp. 123-140, 1981.
6. S.V. Ovchinnikov, Structure of fuzzy binary relations, *Fuzzy Sets and Systems*, Vol. 6, No. 2, pp. 169-195, 1981.
7. B. Bouchon, G. Cohen and P. Frankl, Metrical properties of fuzzy relations, *Problems of Control and Information Theory*, Vol. 11, No. 5, pp. 389-396, 1982.

8. W. Bandler and L.J. Kohout, On-new types of homomorphisms and congruences for partial algebraic structures and n-ary relations, *International Journal of General Systems*, Vol. 12, No. 2, pp. 149-157, 1986.
9. L.A. Zadeh, Fuzzy logic, *IEEE Computer*, Vol. 21, No. 4, pp. 83-93, 1988.
10. U. Hohle, Quotients with respect to similarity relations, *Fuzzy Sets and Systems*, Vol. 27, No. 1, pp. 31-44, 1988.
11. W. Kolodziejczyk, Decomposition problem of fuzzy relations: Further results, *International Journal of General Systems*, Vol. 14, No. 4, pp. 307-315, 1988.
12. Kaufman, A. and M.M. Gupta, Introduction to Fuzzy Arithmetic: Theory and Applications, Van Nostrand Reinhold, NY, 1991.
13. J.C. Fodor, Traces of fuzzy binary relations, *Fuzzy Sets and systems*, Vol. 50, No. 3, pp. 331-341, 1991.
14. J.X. Li, An upper bound on indices of finite fuzzy relations, *Fuzzy Sets and Systems*, Vol. 49, No. 3, pp. 317-321, 1992.
15. B. De Baets and E.E. Kerre, Fuzzy relational compositions, *Fuzzy Sets and Systems*, Vol. 60, No. 1, pp. 109-120, 1993.
16. J. Vrba, General decomposition problem of fuzzy relations, *Fuzzy Sets and Systems*, Vol. 54, No. 1, pp. 69-79, 1993.
17. P. Faurous and J.P. Fillard, A new approach to the similarity in the fuzzy set theory, *Information Sciences*, Vol. 75, No. 3, pp. 213-221, 1993.
18. R. Kruse, J. Gebhardt, and F. Klawon, *Foundations of Fuzzy Systems*, Wiley, Chichester, 1994.
19. J. Mordeson, and C.S. Peng, Operations on fuzzy graphs, *Information Sciences*, Vol. 79, No. 3, pp. 159-170, 1994.
20. Klir, G.J. and B. Yuan, *Fuzzy Sets and Fuzzy Logic*: *Theory and Applications*, Prentice Hall, Upper Saddle River, NJ, 1995.
21. T.J. Ross, *Fuzzy Logic with Engineering Applications*, McGraw-Hill, Inc., New York, NY, pp. 134-146,1995.

CHAPTER 4

Fuzzy Implications

4.1 INTRODUCTION

Let p = "x is in A" and q = "y is in B" are crisp propositions, where A and B are crisp sets for the moment.

The implication $p \rightarrow q$ is interpreted as $\neg (p \wedge \neg q)$. ...(4.1)

"p entails q" means that it can never happen that p is true and q is not true. It is easy to see that

$$p \rightarrow q = \neg p \vee q \quad ...(4.2)$$

The full interpretation of the material implication $p \rightarrow q$ is that the degree of truth of $p \rightarrow q$ quantifies to what extend q is at least as true as p, i.e.

$$p \rightarrow q \text{ is true} \Leftrightarrow \tau(p) \leq \tau(q) \quad ...(4.3)$$

$$p \rightarrow q = \begin{cases} 1 & \text{if } \tau(p) \leq \tau(q) \\ 0 & \text{otherwise} \end{cases}$$

p	q	$p \rightarrow q$
1	1	1
0	1	1
0	0	1
1	0	0

The truth table for the material implication.

Example 4.1: Let p = "x is bigger than 10" and let q = "x is bigger than 9". It is easy to see that $p \rightarrow q$ is true, because it can never happen that x is bigger than 10 and x is not bigger than 9.

This property of material implication can be interpreted as:

if $X \subset Y$ then $X \rightarrow Y$ ...(4.4)

Other interpretation of the implication operator is

$$X \to Y = \sup\{Z | X \cap Z \subset Y\} \qquad ...(4.5)$$

4.2 FUZZY IMPLICATIONS

Consider the implication statement, "if pressure is high then volume is small".

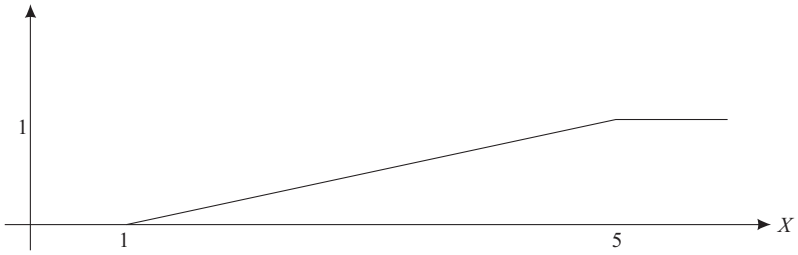

Fig. 4.1 Membership function for "big pressure".

The membership function of the fuzzy set A, big pressure, illustrated in the Fig. 4.1 can be interpreted as

- 1 is in the fuzzy set big pressure with grade of membership 0
- 2 is in the fuzzy set big pressure with grade of membership 0.25
- 4 is in the fuzzy set big pressure with grade of membership 0.75
- x is in the fuzzy set big pressure with grade of membership 1, for all $x \geq 5$

$$A(u) = \begin{cases} 1 & \text{if } u \geq 5 \\ 1 - \dfrac{5-u}{4} & \text{if } 1 \leq u \leq 5 \\ 0 & \text{otherwise} \end{cases} \qquad ...(4.6)$$

The membership function of the fuzzy set B, small volume, can be interpreted as (See Fig. 4.2)

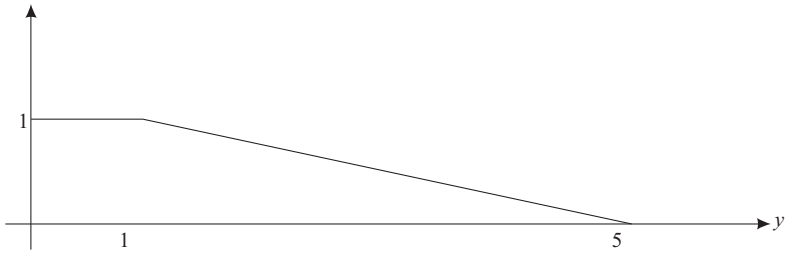

Fig. 4.2 Membership function for "small volume".

- 5 is in the fuzzy set small volume with grade of membership 0
- 4 is in the fuzzy set small volume with grade of membership 0.25
- 2 is in the fuzzy set small volume with grade of membership 0.75
- x is in the fuzzy set small volume with grade of membership 1, for all $x \leq 1$

$$B(v) = \begin{cases} 1 & \text{if } v \geq 1 \\ 1 - \dfrac{v-1}{4} & \text{if } 1 \leq v \leq 5 \\ 0 & \text{otherwise} \end{cases} \qquad ...(4.7)$$

If p is a proposition of the form

x is A

where A is a fuzzy set, for example, big pressure and q is a proposition of the form

y is B

for example, small volume then we define the fuzzy implication $A \to B$ as a fuzzy relation.

It is clear that $(A \to B)(u, v)$ should be defined pointwise and likewise, i.e. $(A \to B)(u, v)$ depends only on $A(u)$ and $B(v)$.

That is

$$(A \to B)(u, v) = I(A(u), B(v)) = A(u) \to B(v) \qquad ...(4.8)$$

In our interpretation $A(u)$ is considered as the truth value of the proposition

"u is big pressure",

and $B(v)$ is considered as the truth value of the proposition

"v is small volume".

that is

$$u \text{ is big pressure} \to v \text{ is small volume} \equiv A(u) \to B(v)$$

Remembering the full interpretation of the material implication

$$p \to q = \begin{cases} 1 & \text{if } \tau(p) \leq \tau(q) \\ 0 & \text{otherwise} \end{cases} \qquad ...(4.9)$$

One possible extension of material implication to implications with intermediate truth values can be

$$A(u) \to B(v) = \begin{cases} 1 & \text{if } \tau(p) \leq \tau(q) \\ 0 & \text{otherwise} \end{cases} \qquad ...(4.10)$$

"4 is big pressure" $\to$ "1 is small volume"

$$A(4) \to B(1) = 0.75 \to 1 = 1$$

However, it is easy to see that this fuzzy implication operator (called Standard Strict) sometimes is not appropriate for real-life applications. Namely, let $A(u) = 0.8$ and $B(v) = 0.8$. Then we have

$$A(u) \rightarrow B(v) = 0.8 \rightarrow 0.8 = 1$$

Suppose there is a small error of measurement in $B(v)$, and instead of 0.8 we have 0.7999. Then

$$A(u) \rightarrow B(v) = 0.8 \rightarrow 0.7999 = 0$$

This example shows that small changes in the input can cause a big deviation in the output, i.e. our system is very sensitive to rounding errors of digital computation and small errors of measurement.

A smoother extension of material implication operator can be derived from the equation

$$X \rightarrow Y = \sup \{Z | X \cap Z \subset Y\} \quad ...(4.11)$$

That is

$$A(u) \rightarrow B(v) = \sup \{z | \min \{A(u), z\} \leq B(v)\} \quad ...(4.12)$$

so,

$$A(u) \rightarrow B(v) = \begin{cases} 1 & \text{if } A(u) \leq B(v) \\ B(v) & \text{otherwise} \end{cases} \quad ...(4.13)$$

This operator is called Godel implication. Other possibility is to extend the original definition,

$$\rightarrow q = \neg p \vee q$$

using the definition of negation and union

$$A(u) \rightarrow B(v) = \max \{1 - A(u), B(v)\} \quad ...(4.14)$$

This operator is called Kleene-Dienes implication.

In many practical applications they use Mamdani's implication operator to model causal relationship between fuzzy variables.

This operator simply takes the minimum of truth values of fuzzy predicates

$$A(u) \rightarrow B(v) = \min \{A(u), B(v)\} \quad ...(4.15)$$

It is easy to see this is not a correct extension of material implications, because $0 \rightarrow 0$ yields zero. However, in knowledge-based systems, we are usually not interested in rules, where the antecedent part is false.

Larsen $x \rightarrow y = xy$ ...(4.16)

Lukasiewiez $x \rightarrow y = \min\{1, 1 - x + y\}$ ...(4.17)

Mamdani $x \rightarrow y = \min\{x, y\}$ ...(4.18)

Standard Strict $$x \rightarrow y = \begin{cases} 1 & \text{if } x \leq y \\ 0 & \text{otherwise} \end{cases} \quad ...(4.19)$$

Godel $$x \rightarrow y = \begin{cases} 1 & \text{if } x \leq y \\ y & \text{otherwise} \end{cases} \quad ...(4.20)$$

Gains $$x \to y = \begin{cases} 1 & \text{if } x \le y \\ y/x & \text{otherwise} \end{cases} \quad ...(4.21)$$

Kleene-Dienes $$x \to y = \max\{1 - x, y\} \quad ...(4.22)$$

Kleene-Dienes-Luk $$x \to y = 1 - x + xy \quad ...(4.23)$$

4.3 MODIFIERS

Let A be a fuzzy set in X. Then we can define the fuzzy sets "very A" and "more or less A" by

$$(\text{very } A)(x) = A(x)^2, (\text{more or less } A)(x) = \sqrt{A(x)} \quad ...(4.24)$$

The use of fuzzy sets provides a basis for a systematic way for the manipulation of vague and imprecise concepts.

In particular, we can employ fuzzy sets to represent linguistic variables.

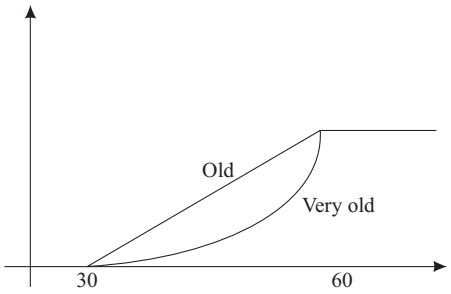

Fig. 4.3 "Very old".

A linguistic variable can be regarded either as a variable whose value is a fuzzy number or as a variable whose values are defined in linguistic terms.

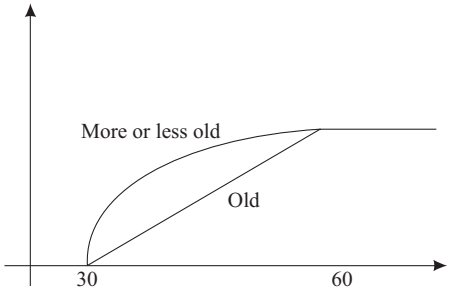

Fig. 4.4 "More or less old".

4.3.1 LINGUISTIC VARIABLES

A linguistic variable is characterized by a quintuple

$$(x, T(x), U, G, M) \qquad ...(4.25)$$

in which

- x is the name of variable;
- $T(x)$ is the term set of x, that is, the set of names of linguistic values of x with each value being a fuzzy number defined on U;
- G is a syntactic rule for generating the names of values of x; and
- M is a semantic rule for associating with each value its meaning.

For example, if speed is interpreted as a linguistic variable, then its term set T (speed) could be

$$T = \{\text{slow, moderate, fast, very slow, more or less fast, ...}\}$$

where each term in T (speed) is characterized by a fuzzy set in a universe of discourse $U = [0, 100]$.

We might interpret "slow" as "a speed below about 40 mph", "moderate" as "a speed close to 55 mph", and "fast" as "a speed above about 70 mph".

These terms can be characterized as fuzzy sets whose membership functions are shown in Fig. 4.5.

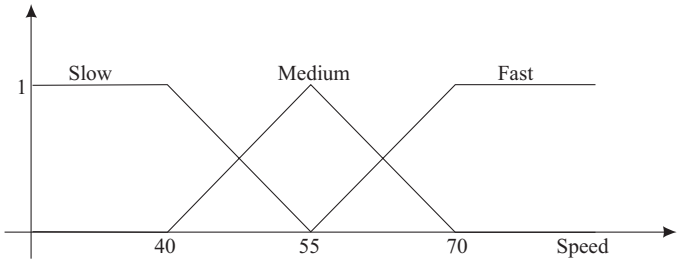

Fig. 4.5 Values of linguistic variable speed.

In many practical applications we normalize the domain of inputs and use the type of fuzzy partition shown in Fig. 4.6.

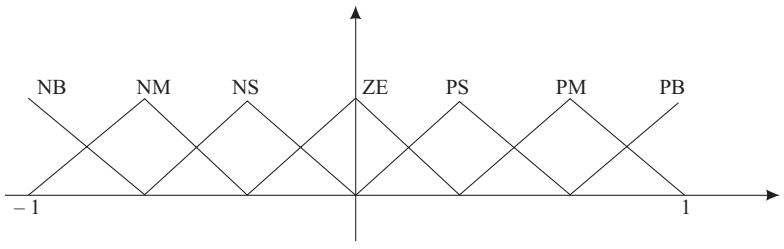

Fig. 4.6 A possible partition of [–1, 1].

Here we used the abbreviations

NB Negative Big,

[*NM*] Negative Medium,

NS Negative Small,

[*ZE*] Zero,

[*PS*] Positive Small,

[*PM*] Positive Medium,

[*PB*] Positive Big.

4.3.2 The Linguistic Variable Truth

Truth = {Absolutely false, Very false, False, Fairly true, True, Very true, Absolutely true}.

One may define the membership function of linguistic terms of truth as

$$\text{True } (u) = u \qquad \text{...(4.26)}$$

for each $u \in [0, 1]$.

$$\text{False } (u) = 1 - u \qquad \text{...(4.27)}$$

for each $u \in [0, 1]$.

$$\text{Absolutely false } (u) = \begin{cases} 1 & \text{if } u = 0 \\ 0 & \text{otherwise} \end{cases} \qquad \text{...(4.28)}$$

$$\text{Absolutely true } (u) = \begin{cases} 1 & \text{if } u = 1 \\ 0 & \text{otherwise} \end{cases} \qquad \text{...(4.29)}$$

The interpolation if absolutely false and absolutely true are shown in Fig. 4.7.

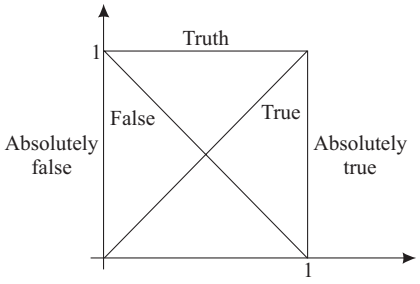

Fig. 4.7 Interpretation of absolutely false and absolutely true.

The word "Fairly" is interpreted as "more or less".

$$\text{Fairly true } (u) = \sqrt{u} \qquad ...(4.30)$$

for each $u \in [0, 1]$.

$$\text{Very true } (u) = u^2 \qquad ....(4.31)$$

for each $u \in [0, 1]$.

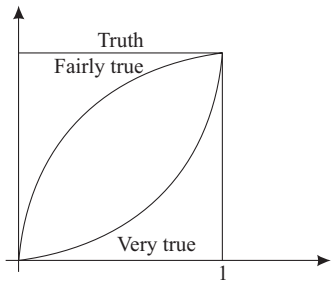

Fig. 4.8 Interpretation of fairly true and very true.

The word "Fairly" is interpreted as "more or less".

$$\text{Fairly false } (u) = \sqrt{1-u} \qquad ...(4.32)$$

for each $u \in [0, 1]$.

$$\text{Very false } (u) = (1-u)^2 \qquad ...(4.33)$$

for each $u \in [0, 1]$.

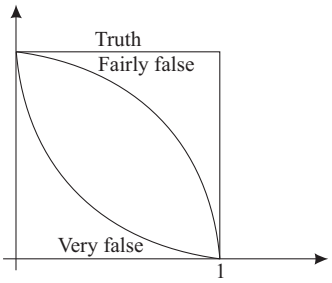

Fig. 4.9 Interpretation of fairly false and very false.

Suppose we have the fuzzy statement "x is A". Let τ be a term of linguistic variable Truth. Then the statement "x is A is τ" is interpreted as "x is τ o A".

Where

$$(\tau \text{ o } A)(u) = \tau(A(u)) \qquad ...(4.34)$$

for each $u \in [0, 1]$.

For example, let τ = "true". Then

"x is A is true"

is defined by

"x is τ o A" = "x is A"

because

$$(\tau \text{ o } A)(u) = \tau(A(u)) = A(u)$$

for each $u \in [0, 1]$.

It is why "everything we write is considered to be true".

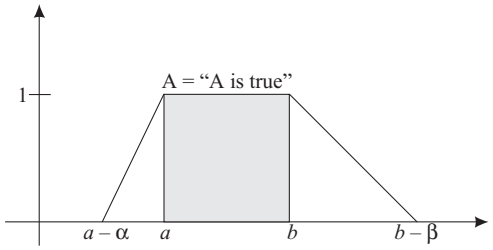

Fig. 4.10 Interpretation of "*A* is true".

Let τ = "absolutely true". Then the statement "x is A is Absolutely true" is defined by "x is τ o A", where

$$(\tau \text{ o } A)(x) = \begin{cases} 1 & \text{if } A(x) = 1 \\ 0 & \text{otherwise} \end{cases} \quad \text{...(4.35)}$$

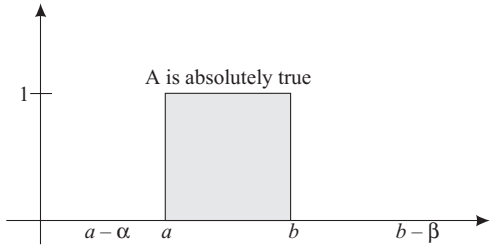

Fig. 4.11 Interpretation of "*A* is absolutely true".

Let τ = "absolutely false". Then the statement "x is A is Absolutely false" is defined by "x is τ o A", where

$$(\tau \text{ o } A)(x) = \begin{cases} 1 & \text{if } A(x) = 0 \\ 0 & \text{otherwise} \end{cases} \quad \text{...(4.36)}$$

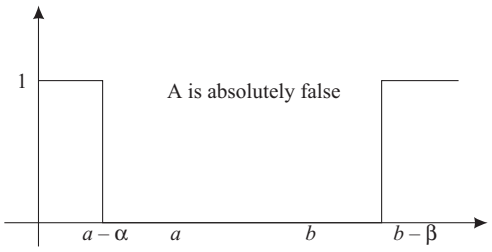

Fig. 4.12 Interpretation of "*A* is absolutely false".

Let τ = "Fairly true". Then the statement "*x* is *A* is Fairly true" is defined by "*x* is τ o *A*", where

$$(\tau \text{ o } A)(x) = \sqrt{A(x)} \qquad ...(4.37)$$

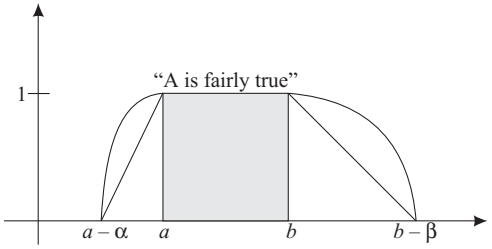

Fig. 4.13 Interpretation of "*A* is fairly true".

Let τ = "Very true". Then the statement "*x* is *A* is Fairly true" is defined by "*x* is τ o *A*", where

$$(\tau \text{ o } A)(x) = (A(x))^2 \qquad ...(4.38)$$

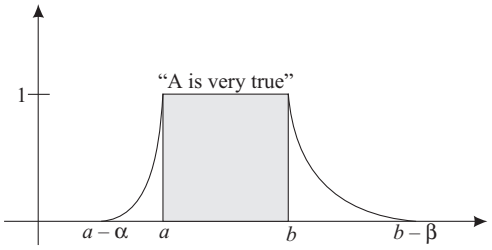

Fig. 4.14 Interpretation of " *A* is very true".

QUESTION BANK

1. What are the fuzzy implications? Explain with examples.
2. What are the fuzzy modifiers? Explain with an example.
3. What are the linguistic variables? Give examples.

4. Explain the linguistic variable TRUTH with examples.
5. Given the set 6 of people in the following age groups:

   0 –10
   10 – 20
   20 – 30
   30 – 40
   40 – 50
   50 – 60
   60 – 70
   70 – 80
   80 and above

   Represent graphically the membership functions of "young", "middle-aged" and "old".

REFERENCES

1. L.A. Zadeh, The concept of a linguistic variable and its application to approximate reasoning, Part 1, *Information Sciences*, Vol. 8, pp. 199-249, 1975.
2. E.H. Mamdani, Advances in the linguistic synthesis of fuzzy controllers, *International Journal of Man-Machine Studies*, Vol. 8, No. 6, pp. 669-678, 1976.
3. E.H. Mamdani, Applications of fuzzy logic to approximate reasoning using linguistic systems, *IEEE Transactions on Systems*, *Man*, *and Cybernetics*, Vol. 26, No. 12, pp. 1182-1191, 1977.
4. J.F. Baldwin and B.W. Pilsworth, Axiomatic approach to implication for approximate reasoning with fuzzy logic, *Fuzzy Sets and Systems*, Vol. 3, No. 2, pp. 193-219, 1980.
5. W. Bandler and L.J. Kohout, Fuzzy power sets and fuzzy implication operators, *Fuzzy Sets and Systems*, Vol. 4, No. 1, pp. 13-30, 1980.
6. W. Bandler, and L.J. Kohout, Semantics of implication operators and fuzzy relational products, *International Journal of Man-Machine Studies*, Vol. 12, No. 1, pp. 89-116, 1980.
7. R. Willmott, Two fuzzier implication operators in the theory of fuzzy power sets, *Fuzzy Sets and Systems*, Vol. 4, No. 1, pp. 31-36, 1980.
8. S. Weber, A general concept of fuzzy connectives, negations and implications based on t-norms and t-conorms, *Fuzzy Sets and Systems*, Vol. 11, No. 2, pp. 115-134, 1983.
9. D. Dubois and H. Prade, A theorem on implication functions defined from triangular norms, *Stochastica*, Vol. 8, No. 3, pp. 267-279, 1984.
10. E. Trillas and L. Valverde, On mode and implications in approximate reasoning, In: M. M. Gupta, A. Kandel, W. Bandler and J.B. Kisska [Eds.], *Approximate Reasoning in Expert Systems*, North-Holland, New York, pp. 157-166, 1985.
11. J.E. Ahlquist, Application of fuzzy implication to probe nonsymmetric relations: Part 1, *Fuzzy Sets and Systems*, Vol. 22, No. 3, pp. 229-244, 1987.
12. K.W. Oh and W. Bandler, Properties of fuzzy implication operators, *International Journal of Approximate Reasoning*, Vol. 1, No. 3, pp. 273-285, 1987.

13. P. Smets and P. Magrez, Implication in fuzzy logic, *International Journal of Approximate Reasoning*, Vol. 1, No. 4, pp. 327-347, 1987.
14. P. Smets and P. Magrez, The measure of the degree of truth and the grade of membership, *Fuzzy Sets and Systems*, Vol. 25, No. 1, pp. 67-72, 1988.
15. Z. Cao and A. Kandel, *Applicability of Some Fuzzy Implication Operators*, Vol. 31, No. 2, pp. 151-186, 1989.
16. R. Da, E.E. Kerre, G. De Cooman, B. Cappelle and F. Vanmassenhove, Influence of the fuzzy implication operator on the method-of-cases inference rule, *International Journal of Approximate Reasoning*, Vol. 4, No. 4, pp. 307-318, 1990.
17. J.C. Fodor, On fuzzy implication operators, *Fuzzy Sets and Systems*, Vol. 42, No. 3, pp. 293-300, 1991.
18. A. Piskunov, Fuzzy implication in fuzzy systems control, *Fuzzy Sets and Systems*, Vol. 45, No. 1, pp. 25-35, 1992.
19. D. Ruan and E.E Kerre, Fuzzy implication operators and generalized fuzzy method of cases, *Fuzzy Sets and Systems*, Vol. 54, No. 1, pp. 23-37, 1993.
20. J.L. Castro, M. Delgado and E. Trillas, Inducing implication relations, *International Journal of Approximate Reasoning*, Vol. 10, No. 3, pp. 235-250, 1994.
21. W.M. Wu, Commutative implications on complete lattices, *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, Vol. 2, No. 3, pp. 333-341, 1994.

CHAPTER 5

The Theory of Approximate Reasoning

5.1 INTRODUCTION

In 1975 Zadeh introduced the theory of approximate reasoning. This theory provides a powerful framework for reasoning in the face of imprecise and uncertain information. Central to this theory is the representation of propositions as statements assigning fuzzy sets as values to variables.

Suppose we have two interactive variables $x \in X$ and $y \in Y$ and the causal relationship between x and y is completely known. Namely, we know that y is a function of x

$$y = f(x)$$

Then we can make inferences easily

Premise	$y = f(x)$
Fact	$x = x^1$
Consequence	$y = f(x^1)$

This inference rule says that if we have $y = f(x)$, $\forall x \in X$ and we observe that $x = x^1$, then y takes the value $f(x^1)$.

More often than not we do not know the complete causal link f between x and y, only we know the values of $f(x)$ for some particular values of x

$\mathfrak{R}_1$: If $x = x_1$ then $y = y_1$

also

$\mathfrak{R}_2$: If $x = x_2$ then $y = y_2$

also

…

also

$\Re_n$: If $x = x_n$ then $y = y_n$

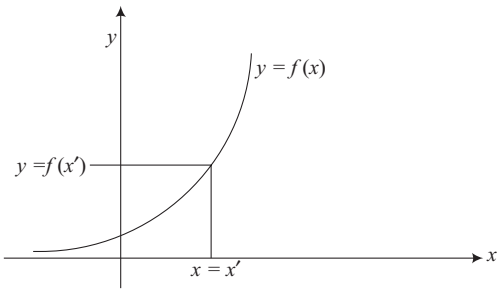

Fig. 5.1 Simple crisp inference.

Suppose that we are given an $x^1 \in X$ and want to find an $y^1 \in Y$ which corresponds to x^1 under the rule-base.

$\Re_1$: If $x = x_1$ then $y = y_1$

also

$\Re_2$: If $x = x_2$ then $y = y_2$

also

…

also

$\Re_n$: If $x = x_n$ then $y = y_n$

fact: $x = x^1$

Consequence: $y = y^1$

This problem is frequently quoted as interpolation.

Let x and y be linguistic variables, e.g. "x is high" and "y is small". The basic problem of approximate reasoning is to find the membership function of the consequence C from the rule-base $\{\Re_1, \ldots, \Re_n\}$ and the fact A.

$\Re_1$: If x is A_1 then y is C_1

also

$\Re_2$: If x is A_2 then y is C_2

also	
…	
also	
$\Re_n$:	If x is A_n then y is C_n
fact:	x is A
	Consequence: y is c

Zadeh introduced a number of translation rules, which allow us to represent some common linguistic statements in terms of propositions in our language.

5.2 TRANSLATION RULES

5.2.1 Entailment Rule

x is A	Menaka is very young
$A \subset B$	very young $\subset$ young
x is B	Menaka is young

5.2.2 Conjunction Rule

x is A
x is B
---
x is $A \cap B$

Temperature is not very high
Temperature is not very low

Temperature is not very high and not very low

5.2.3 Disjunction Rule

x is A
or x is B
---
x is $A \cup B$

Temperature is not very high
or Temperature is not very low

Temperature is not very high or not very low

5.2.4 Projection Rule

(x, y) have relation R

---

x is $\prod_x (R)$

(x, y) have relation R

---

y is $\prod_y (R)$

(x, y) is close to (3, 2)

---

x is close to 3

(x, y) is close to (3, 2)

---

y is close to 2

5.2.5 Negation Rule

not (x is A)

---

x is $\neg A$

not (x is high)

---

x is not high

In fuzzy logic and approximate reasoning, the most important fuzzy implication inference rule is the Generalized Modus Ponens (GMP). The classical Modus Ponens inference rule says:

premise	if p then q
fact	p
consequence:	q

This inference rule can be interpreted as: If p is true and $p \rightarrow q$ is true then q is true.

The fuzzy implication inference is based on the compositional rule of inference for approximate reasoning suggested by Zadeh.

5.2.6 Compositional Rule Of Inference

premise	if x is A then y is B
fact	x is A^1
consequence:	y is B^1

where the consequence B^1 is determined as a composition of the fact and the fuzzy implication operator.

$$B^1 = A^1 \text{ o } (A \rightarrow B) \qquad ...(5.1)$$

that is,

$$B'(v) = \sup_{u \in U} \min \{A'(u), (A \rightarrow B)(u, v)\}, v \in V \qquad ...(5.2)$$

The consequence B^1 is nothing else but the shadow of $A \rightarrow B$ on A^1.

The Generalized Modus Ponens, which reduces to calssical modus ponens when $A^1 = A$ and $B^1 = B$, is closely related to the forward data-driven inference which is particularly useful in the Fuzzy Logic Control.

The classical Modus Tollens inference rule says: If $p \rightarrow q$ is true and q is false then p is false. The Generalized Modus Tollens,

premise	if x is A then y is B
fact	y is B^1
consequence:	x is A^1

which reduces to "Modus Tollens" when $B = \neg B$ and $A^1 = \neg A$, is closely related to the backward goal-driven inference which is commonly used in expert systems, especially in the realm of medical diagnosis.

5.3 RATIONAL PROPERTIES

Suppose that A, B and A^1 are fuzzy numbers. The Generalized Modus Ponens should satisfy some rational properties.

5.3.1 Basic Property

if x is A then y is B

x is A

---

y is B

if pressure is big	then volume is small
pressure is big	
	volume is small

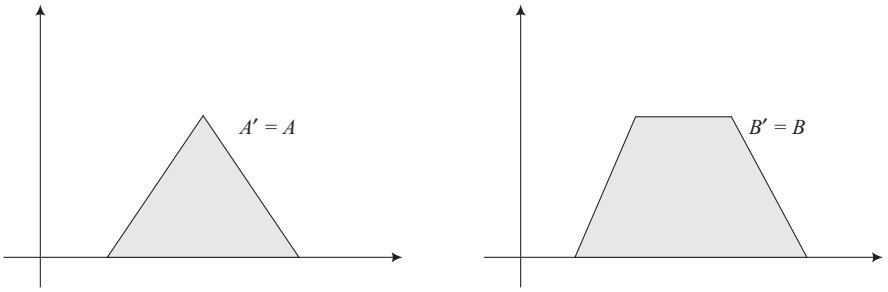

Fig. 5.2 Basic property.

5.3.2 Total Indeterminance

if x is A then y is B
x is $\neg A$

---

y is unknown

if pressure is big then volume is small
pressure is not big

---

volume is unknown

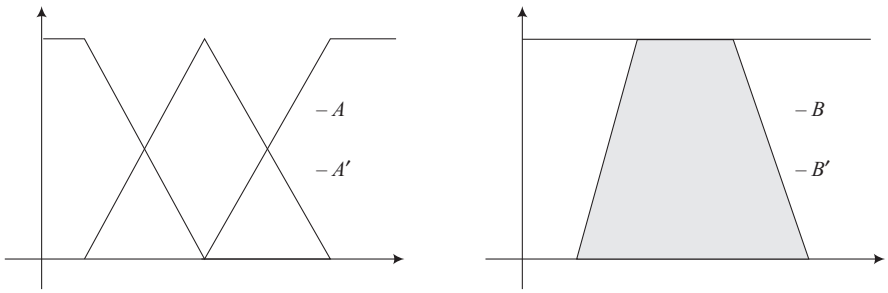

Fig. 5.3 Total indeterminance.

5.3.3 Subset

if x is A then y is B
x is $A^1 \subset A$

---

y is B

if pressure is big then volume is small
pressure is very big

volume is small

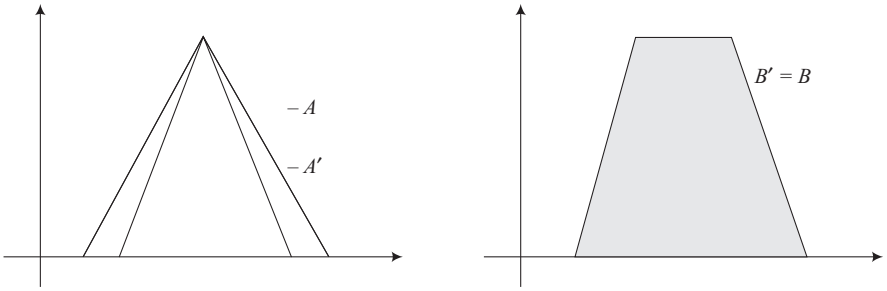

Fig. 5.4 Subset property.

5.3.4 Superset

if x is A then y is B
x is A^1

y is $B^1 \supset B$

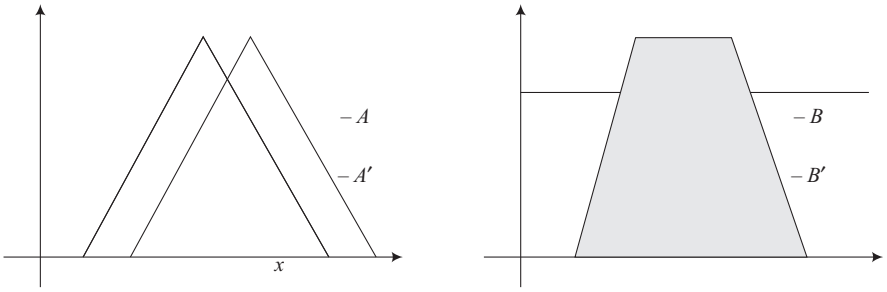

Fig. 5.5 Superset property.

Suppose that A, B and A^1 are fuzzy numbers. We show that the Generalized Modus Ponens with Mamdani's implication operator does not satisfy all the four properties listed above.

Example 5.1: (The GMP with Mamdani implication)

if x is A then y is B
x is A^1

y is B^1

where the membership function of the consequence B^1 is defined by

$$B'(y) = \sup \{A'(x) \wedge A(x) \wedge B(y) \mid x \in R\}, y \in R$$

Basic property: Let $A^1 = A$ and let $y \in R$ be arbitrarily fixed. Then we have

$$B'(y) = \sup_x \min \{A(x), \min \{A(x), B(y)\}\}$$

$$= \sup_x \min \{A(x), B(y)\}$$

$$= \sup_x \min \left\{B(y), \sup_x A(x)\right\}$$

$$= \min \{B(y), 1\}$$

$$= B(y)$$

So the basic property is satisfied.

Total indeterminance: Let $A^1 = \neg A = 1 - A$ and let $y \in R$ be arbitrarily fixed. Then we have

$$B'(y) = \sup_x \min \{1 - A(x), \min \{A(x), B(y)\}\}$$

$$= \sup_x \min \{A(x), 1 - A(x), B(y)\}$$

$$= \min \left\{B(y), \sup_x \min \{A(x), 1 - A(x),\}\right\}$$

$$= \min \{B(y), 1/2\}$$

$$= 1/2 \; B(y) < 1$$

This means that the total indeterminance property is not satisfied.

Subset: Let $A' \subset A$ and let $y \in R$ be arbitrarily fixed. Then we have

$$B'(y) = \sup_x \min \{A'(x), \min \{A(x), B(y)\}\}$$

$$= \sup_x \min \{A(x), A'(x), B(y)\}$$

$$= \min \left\{B(y), \sup_x A'(x)\right\}$$

$$= \min \{B(y), 1\}$$

$$= B(y)$$

So the subset is satisfied.

Superset: Let $y \in R$ be arbitrarily fixed. Then we have

$$B'(y) = \sup_x \min \{A'(x), \min \{A(x), B(y)\}\}$$

$$= \sup_x \min \{A(x), A'(x), B(y)\} \leq B(y)$$

So the superset property of GMP is not satisfied by Mamdani's implication operator.

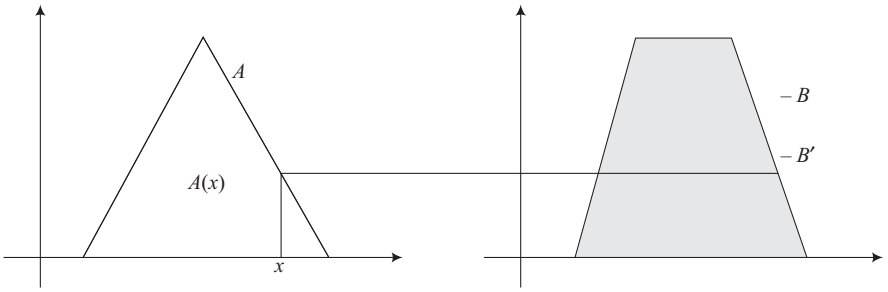

Fig. 5.6 The GMP with Mamdani's implication operator.

Example 5.2: (The GMP with Larsen's product implication)

$$\frac{\begin{array}{l}\text{if } x \text{ is } A \text{ then } y \text{ is } B \\ x \text{ is } A^1\end{array}}{y \text{ is } B^1}$$

where the membership function of the consequence B^1 is defined by

$$B'(y) = \sup_x \min \{A'(x), A(x)\, B(y) \mid x \in R\}\ y \in R$$

Basic property: Let $A^1 = A$ and let $y \in R$ be arbitrarily fixed. Then we have

$$B'(y) = \sup_x \min \{A(x), A(x)\, B(y)\} = B(y)$$

So the basic property is satisfied.

Total indeterminance: Let $A^1 = \neg A = 1 - A$ and let $y \in R$ be arbitrarily fixed. Then we have

$$B'(y) = \sup_x \min \{1 - A(x), A(x)\, B(y)\}$$

$$= \frac{B(y)}{1 + B(y)} < 1$$

This means that the total indeterminance property is not satisfied.

Subset: Let $A' \subset A$ and let $y \in R$ be arbitrarily fixed. Then we have

$$B'(y) = \sup_x \min \{A'(x), A(x)\, B(y)\}$$

$$= \sup_x \min \{A(x), A'(x)\, B(y)\}$$

$$= B(y)$$

So the subset property is satisfied.

Superset: Let $y \in R$ be arbitrarily fixed. Then we have

$$B'(y) = \sup_x \min \{A'(x), A(x)\, B(y)\} \leq B(y)$$

So, the superset property is not satisfied.

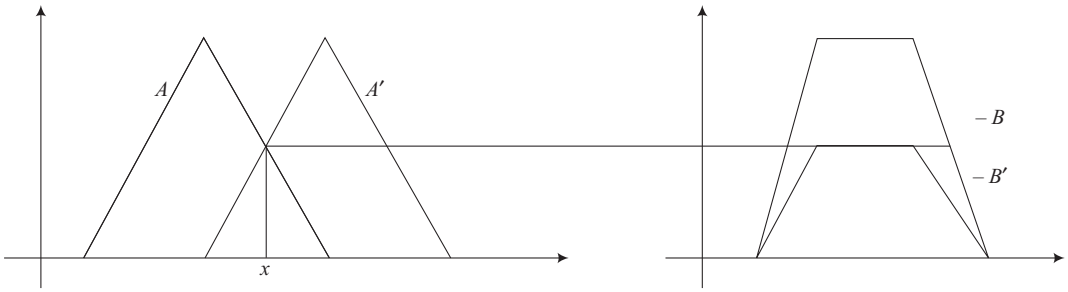

Fig. 5.7 The GMP with Larsen's implication operator.

QUESTION BANK

1. Explain the theory of approximate reasoning.
2. What are the translation rules? Explain them with examples.
3. What are the rational properties? Explain them.
4. Explain generalized modus ponens with Mamdani's implication.
5. Explain generalized modus ponens with Larsen's implication.
6. Given

   $C \vee D$

   $\sim H \Rightarrow (A \wedge \sim B)$

   $C \vee D \Rightarrow \sim H$

   $(A \wedge \sim B) \Rightarrow (R \vee S)$

   Can $(R \vee S)$ be inferred from the above?

REFERENCES

1. L.A. Zadeh, Fuzzy logic and approximate reasoning, *Syntheses*, Vol. 30, No. 1, pp. 407-428, 1975.
2. L.A. Zadeh, The concept of a linguistic variable and its application to approximate reasoning-Part I, *Information Sciences*, Vol. 8, pp. 199-251, 1975.
3. L.A. Zadeh, The concept of a linguistic variable and its application to approximate reasoning-Part II, *Information Sciences*, Vol. 8, pp. 301-357, 1975.
4. L.A. Zadeh, The concept of a linguistic variable and its application to approximate reasoning-Part III, *Information Sciences*, Vol. 9, pp. 43-80, 1975.
5. B.R. Gainess, Foundations of fuzzy reasoning, *International Journal of Man-Machine Studies*, Vol. 8, No. 6, pp. 623-668, 1976.
6. E.H. Mamdani, Application of fuzzy logic to approximate reasoning using linguistic systems, *IEEE Transactions on Systems*, *Man and Cybernetics*, Vol. 26, No. 12, pp. 1182-1191, 1977.
7. J.F. Baldwin, A new approach to approximate reasoning using a fuzzy logic, *Fuzzy Sets and Systems*, Vol. 2, No. 4, pp. 309-325, 1979.
8. M. Mizumoto and H.J. Zimmermann, Comparison of fuzzy reasoning methods, *Fuzzy Sets and Systems*, Vol. 2, No. 4, pp. 309-325, 1979.
9. R. Giles, Semantics for fuzzy reasoning, *International Journal of Man-Machine Studies*, Vol. 17, No. 4, pp. 401-415, 1982.
10. L.A. Zadeh, Linguistic variables, approximate reasoning and dispositions, Medical Information, Vol. 8, pp. 173-186, 1983.
11. L.A. Zadeh, The role of fuzzy logic in the management of uncertainty in expert systems, *Fuzzy Sets and Systems*, Vol. 11, No. 3, pp. 199-228, 1983.
12. M. Sugeno and T. Takagi, *Multi-dimensional Fuzzy Reasoning*, *Fuzzy Sets and Systems*, Vol. 9, No. 3, pp. 313-325, 1983.
13. L.A. Zadeh, Syllogistic reasoning in fuzzy logic and its application to usually and reasoning with dispositions, *IEEE Transactions on Systems*, *Man and Cybernetics*, Vol. 15, No. 6, pp. 754-765, 1985.
14. H. Prade, A computational approach to approximate and plausible reasoning with applications to expert systems, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 7, No. 3, pp. 260-283, 1985.
15. R.R. Yager, Strong truth and rules of inference in fuzzy logic and approximate reasoning, *Cybernetics and Systems*, Vol. 16, No. 1, pp. 23-63, 1985.
16. H. Farreny and H. Prade, Default and inexact reasoning with possibility degress, *IEEE Transactions on Systems*, *Man and Cybernetics*, Vol. 16, No. 2, pp. 270-276, 1986.
17. W.M. Wu, Fuzzy reasoning and fuzzy relational equations, *Fuzzy Sets and Systems*, Vol. 20, No. 1, pp. 67-78, 1986.
18. L.A. Zadeh, A computational theory of dispositions, *International Journal of Intelligent Systems*, Vol. 2, No. 1, pp. 39-63, 1987.

19. M.B. Gorzalczany, A method of inference in approximate reasoning based on interval-valued fuzzy sets, *Fuzzy Sets and Systems*, Vol. 21, No. 1, pp. 1-17, 1987.
20. N.S. Lee, Y.L Grize and K. Dehnad, Qualitative models for reasoning under uncertainty in knowledge–based expert systems, *International Journal of Intelligent Systems*, Vol. 2, No. 1, pp. 15-38, 1987.
21. I.B. Turksen, Approximate reasoning for production planning, *Fuzzy Sets and Systems*, Vol. 26, No. 1, pp. 23-37, 1988.
22. M.S. Ying, Some notes on multi-dimensional fuzzy reasoning, *Cybernetics and Systems*, Vol. 19, No. 4, pp. 281-293, 1988.
23. Z. Cao, A. Kandel and L. Li, A new model of fuzzy reasoning, *Fuzzy Sets and Systems*, Vol. 36, No. 3, pp. 311-325, 1989.
24. P. Magrez and P. Smets, Fuzzy modus ponens: A new model suitable for applications in knowledge-based systems, *International Journal of Intelligent Systems*, Vol. 4, No. 4, pp. 181-200, 1989.
25. P. Torasso and L. Console, approximate reasoning and prototypical knowledge, *International Journal of Approximate Reasoning*, Vol. 3, No. 2, pp. 157-178, 1989.
26. R. Kruse and E. Schwecke, Fuzzy reasoning in a multi-dimensional space of hypothesis, *International Journal of Approximate Reasoning*, Vol. 4, No. 1, pp. 47-68, 1990.
27. C.Z. Luo and Z.P. Wang, Representation of compositional relations in fuzzy reasoning, *Fuzzy Sets and Systems*, Vol. 36, No. 1, pp. 77-81, 1990.
28. C. Feng, Quantitative evaluation of university teaching quality – An application of fuzzy set and approximate reasoning, *Fuzzy Sets and Systems*, Vol. 37, No. 1, pp. 1-11, 1990.
29. J.M. Keller, D, Subhangkasen, K. Unklesbay and N. Unklesbay, An approximate reasoning technique for recognition in color images of beef steaks, *International Journal of General Systems*, Vol. 16, No. 4, pp. 331-342, 1990.
30. R. Lopez de Mantaras, *Approximate Reasoning Models*, Ellis Horwood, Chischster, 1990.
31. D. Dubois and H. Prade, Fuzzy sets in approximate reasoning, Part 1: Inference with Possibility distributions, *Fuzzy Sets and Systems*, Vol. 40, No. 1, pp. 143-202, 1991.
32. D. Dubois and H. Prade, Fuzzy sets in approximate reasoning, Part 2: Logical approaches, *Fuzzy Sets and Systems*, Vol. 40, No. 1, pp. 202-244, 1991.
33. M.M. Gupta and J. Qi, Theory of t-norms and fuzzy inference methods, *Fuzzy Sets and Systems*, Vol. 40, No. 3, pp. 431-450, 1991.
34. S. Dutta, approximate spatial reasoning: Integrating qualitative and quantitative constraints, *International Journal of Approximate Reasoning*, Vol. 5, No. 3, pp. 307-330, 1991.
35. D.G. Schwartz, *A system for Reasoning with Imprecise Linguistic Information*, Vol. 5, No. 5, pp. 463-488, 1991.
36. X. T. Peng, A. Kandel and P.Z. Wang, Concepts, rules and fuzzy reasoning: A factor space approach, *IEEE Transactions on Systems*, *Man and Cybernetics*, Vol. 21, No. 1, pp. 194-205, 1991.
37. E.H. Ruspini, Approximate reasoning: past, present, future, *Information Sciences*, Vol. 57, pp. 297-317, 1991.

38. D.L. Hudson, M.E. Cohen and M.F. Anderson, Approximate reasoning with IF-THEN-UNLESS rule in a medical expert system, *International Journal of Intelligent Systems*, Vol. 7, No. 1, pp. 71-79, 1992.
39. K. Lano, Formal frameworks for approximate reasoning, *Fuzzy Sets and Systems*, Vol. 51, No. 2, pp. 131-146, 1992.
40. S.M. Chen, An improved algorithm for inexact reasoning based on extended fuzzy production rules, *Cybernetics and Systems*, Vol. 23, No. 5, pp. 463-481, 1992.
41. S. Raha and K.S. Ray, Analogy between approximate reasoning and the method of interpolation, *Fuzzy Sets and Systems*, Vol. 51, No. 3, pp. 259-266, 1992.
42. P.V. Reddy and M.S, Babu, Some methods of reasoning for fuzzy conditional propositions, *Fuzzy Sets and Systems*, Vol. 52, No. 3, pp. 229-250, 1992.
43. H.L. Larsen and R.R. Yager, The use of fuzzy relational thesauri for classificatory problem solving in information retrieval and expert systems, *IEEE Transactions*, on *Systems*, *Man and Cybernetics*, Vol. 23, No. 1, pp. 31-41, 1993.
44. V.A. Niskanen, Metric truth as a basis for fuzzy linguistic reasoning, *Fuzzy Sets and Systems*, Vol. 57, No. 1, pp. 1-25, 1993.
45. L.T. Koczy and K. Hirota, Approximate reasoning by linear rule interpolation and general approximation, *International Journal of Approximate Reasoning*, Vol. 9, No. 3, pp. 197-225, 1993.
46. Z. Zhang, An approximate reasoning system: Design and implementation, International journal of approximate reasoning, Vol. 9, No. 4, pp. 315-326, 1993.
47. Z. Bien and M.G. Chun, An inference network for bi-directional approximate reasoning based on an equality measure, *IEEE Transactions on Fuzzy Systems*, Vol. 2, No. 2, pp. 177-180, 1994.
48. Q. Zhao and B. Li, An approximate reasoning system, *International Journal of Pattern Recognition and Artificial Intelligence*, Vol. 7, No. 3, pp. 431-440, 1993.
49. J. Kacprzyk, On measuring the specificity of IF-THEN rules, *International Journal of Approximate Reasoning*, Vol. 11, No. 1, pp. 29-53, 1994.
50. C. Elbert, *Rule-based Fuzzy Classification for Software Quality Control*, Vol. 63, No. 3, pp. 349-358, 1994.

CHAPTER 6

Fuzzy Rule-Based Systems

6.1 INTRODUCTION

Triangular norms were introduced by Schweizer and Sklar to model the distances in probabilistic metric spaces. Functions that qualify as fuzzy intersections and fuzzy unions are usually referred to in the literature as *t*-norms and *t*-conorms, respectively. Furthermore, the standard fuzzy intersection (min operator) produces for any given fuzzy sets the largest fuzzy set from among those produced by all possible fuzzy intersections (*t*-norms). The standard fuzzy union (max operator) produces, on the contrary, the smallest fuzzy set among the fuzzy sets produced by all possible fuzzy unions (*t*-conorms). That is, the standard fuzzy operations occupy specific positions in the whole spectrum of fuzzy operations: the standard fuzzy intersection is the weakest fuzzy intersection, while the standard fuzzy union is the strongest fuzzy union. In fuzzy sets theory triangular norm are extensively used to model the logical connective **and**.

6.2 TRIANGULAR NORM

A mapping T: $[0, 1] \times [0, 1] \rightarrow [0, 1]$
is a triangular norm (*t*-norm for short) if it is symmetric, associative, non-decreasing in each argument and $T(a, 1) = a$, for all $a \in [0, 1]$. In other words, any *t*-norm T satisfies the properties:

Symmetricity : $T(x, y) = T(y, x), \forall x, y \in [0, 1]$ ...(6.1)

Associativity : $T(x, T(y, z)) = T(T(x, y), z), \forall x, y, z \in [0, 1]$ ...(6.2)

Monotonicity : $T(x, y) \leq T(x', y')$ if $x \leq x'$ and $y \leq y'$. ...(6.3)

One identy : $T(x, 1) = x, \forall x \in [0, 1]$ ...(6.4)

These axioms attempt to capture the basic properties of set intersection. The basic *t*-norms are:

- minimum : $\min(a, b) = \min\{a, b\}$ ...(6.5)
- £ukasiewicz: $T_L(a, b) = \max\{a + b.1, 0\}$ ...(6.6)

- product : $TP(a, b) = ab$ ...(6.7)

- weak : $$T_w(a, b) = \begin{cases} \min\{a, b\} & \text{if} \quad \max\{a, b\} = 1 \\ 0 & \text{otherwise} \end{cases}$$ ...(6.8)

- Hamacher : $$H_\gamma(a, b) = \frac{ab}{\gamma + (1-\gamma)(a + b - ab)}, \quad \gamma \geq 0$$ ...(6.9)

- Dubois and Prade : $$D_\alpha(a, b) = \frac{ab}{\max\{a, b, \alpha\}}, \quad \alpha \in (0, 1)$$ ...(6.10)

- Yager : $$Yp(a, b) = 1 - \min\left\{1, \sqrt[p]{[(1-a)^p + (1-b)^p]}\right\}, \quad p > 0$$ ...(6.11)

- Frank : $$F_\lambda(a, b) = \begin{cases} \min\{a, b\} & \text{if} \quad \lambda = 0 \\ T_P(a, b) & \text{if} \quad \lambda = 1 \\ T_L(a, b) & \text{if} \quad \lambda = \infty \\ 1 - \log_\lambda\left[1 + \dfrac{(\lambda^a - 1)(\lambda^b - 1)}{\lambda - 1}\right] & \text{otherwise} \end{cases}$$ ...(6.12)

All *t*-norms may be extended, through associativity, to $n > 2$ arguments. The minimum *t*-norm is automatically extended and

$$T_P(a_1, a_2, ..., a_n) = a_1 X a_2 X ... X a_n$$ ...(6.13)

$$T_L(a_1, a_2, ..., a_n) = \max\left\{\sum_{i=1}^{n} a_i - n + 1, 0\right\}$$ ...(6.14)

A *t*-norm *T* is called strict if *T* is strictly increasing in each argument. Triangular co-norms are extensively used to model logical connectives **or**.

6.3 TRIANGULAR CONORM

A mapping $S : [0, 1] \times [0, 1] \rightarrow [0, 1]$,
is a triangular co-norm (*t*-conorm) if it is symmetric, associative, non-decreasing in each argument and $S(a, 0) = a$, for all $a \in [0, 1]$. In other words, any *t*-conorm *S* satisfies the properties:

- Symmetry : $S(x, y) = S(y, x)$ ...(6.15)
- Associativity : $S(x, S(y, z)) = S(S(x, y), z)$ ...(6.16)
- Monotonicity : $S(x, y) \leq S(x', y')$ if $x \leq x'$ and $y \leq y'$ ...(6.17)
- Zero identity : $S(x, 0) = x, \forall x \in [0, 1]$ ...(6.18)

If T is a t-norm then the equality

$$S(a, b) = 1 - T(1 - a, 1 - b) \quad ...(6.19)$$

defines a t-conorm and we say that S is derived from T. The basic t-conorms are:

- maximum : $\max(a, b) = \max\{a, b\}$ ...(6.20)
- Łukasiewicz : $S_L(a, b) = \min\{a + b, 1\}$ ...(6.21)
- probabilistic : $S_P(a, b) = a + b.ab$ ...(6.22)
- strong : $\text{STRONG}(a, b) = \begin{cases} \max\{a, b\} & \text{if } \min(a, b) = 0 \\ 1 & \text{otherwise} \end{cases}$ ...(6.23)
- Hamacher : $\text{HOR}_\gamma(a, b) = \dfrac{a + b - (2 - \gamma)ab}{1 - (1 - \gamma)ab}, \quad \gamma \geq 0$ ...(6.24)
- Yager : $\text{YOR}_P(a, b) = \min\left\{1, \sqrt[p]{a^p + b^p}\right\}, \quad P > 0$ ...(6.25)

Lemma 6.1: Let T be a t-norm. Then the following statement holds

$$T_w(x, y) \leq T(x, y) \leq \min\{x, y\}, \forall x, y \in [0, 1]$$

Proof: From monotonicity, symmetricity and the external condition we get

$$T(x, y) \leq T(x, 1) \leq x$$

$$T(x, y) = T(y, x) \leq T(y, 1) \leq y$$

This means that $T(x, y) \leq \min\{x, y\}$.

Lemma 6.2: Let S be a t-conorm. Then the following statement holds

$$\max\{a, b\} \leq s(a, b) \leq STRONG(a, b), \forall a, b \in [0,1]$$

Proof: From monotonicity, symmetricity and the extremal condition we get

$$S(x, y) \geq S(x, 0) \geq x$$

$$S(x, y) \geq S(y, x) \geq S(y, 0) \geq y$$

This means that $S(x, y) \geq \max\{x, y\}$.

Lemma 6.3: $T(a, a) = a$ holds for any $a \in [0, 1]$ if and only if T is the minimum norm.

Proof: If $T(a, b) = \min(a, b)$ then $T(a, a) = a$ holds obviously. Suppose $T(a, a) = a$ for any $a \in [0, 1]$, and $a \leq b \leq 1$. We can obtain the following expression using monotonicity of T

$$a = T(a, a) \leq T(a, b) \leq \min\{a, b\}.$$

From commutativity of T it follows that

$$a = T(a, a) \leq T(b, a) \leq \min\{b, a\}.$$

These equations show that $T(a, b) = \min\{a, b\}$ for any $a \in [0, 1]$.

Lemma 6.4: The distributive law of *t*-norm *T* on the max operator holds for any $a, b, c \in [0, 1]$.

$$T(\max\{a, b\}, c) = \max\{T(a, c), T(b, c)\}.$$

6.4 *t*-NORM-BASED INTERSECTION

Let *T* be a *t*-norm. The *T*-intersection of *A* and *B* is defined as

$$(A \cap B)(t) - T(A(t), B(t)) \quad ...(6.26)$$

for all $t \in X$,

Example 6. 1: Let $T(x, y) = \angle AND(x, y) = \{x + y - 1, 0\}$

be the £ukasiewicz *t*-norm. Then we have

$$(A \cap B)(t) = \max\{A(t) + B(t - 1, 0)\}$$

for all $t \in X$.

Let *A* and *B* be fuzzy subsets of $X = \{x_1, x_2, x_3, x_4, x_5, x_6, x_7\}$ and be defined by

$$A = 0.0/x_1 + 0.3/x_2 + 0.6/x_3 + 1.0/x_4 + 0.6/x_5 + 0.3/x_6 + 0.0/x_7$$

$$B = 0.1/x_1 + 0.3/x_2 + 0.9/x_3 + 1.0/x_4 + 1.0/x_5 + 0.3/x_6 + 0.2/x_7.$$

Then $A \cap B$ has the following form

$$A \cap B = 0.0/x_1 + 0.0/x_2 + 0.5/x_3 + 1.0/x_4 + 0.6/x_5 + 0.0/x_6 + 0.2/x_7.$$

The operation union can be defined by the help of triangular conorms.

6.5 *t*-CONORM-BASED UNION

Let *S* be a *t*-conorm. The *S*-union of *A* and *B* is defined as

$$(A \cap B)(t) = S(A(t), B(t)) \quad ...(6.27)$$

for all $t \in X$.

Example 6. 2: Let (S(x, y) = LOR (x, y) = min {x + y, 1}be the £ukasiewicz *t*-conorm.

Then we have

$$(A \cup B)(t) = \min\{A(t), B(t)1\} \quad ...(6.28)$$

for all $t \in X$.

Let *A* and *B* be fuzzy subsets of $X = \{x_1, x_2, x_3, x_4, x_5, x_6, x_7\}$ and be defined by

$$A = 0.0/x_1 + 0.3/x_2 + 0.6/x_3 + 1.0/x_4 + 0.6/x_5 + 0.3/x_6 + 0.0/x_7$$

$$B = 0.1/x_1 + 0.3/x_2 + 0.9/x_3 + 1.0/x_4 + 1.0/x_5 + 0.3/x_6 + 0.2/x_7$$

Then $A \cup B$ has the following form

$$A \cup B = 0.1/x_1 + 0.6/x_2 + 1.0/x_3 + 1.0/x_4 + 1.0/x_5 + 0.6/x_6 + 0.2/x_7.$$

If we are given an operator *C* such that

$$\min\{a, b\} \le C(a, b) \le \max\{a, b\}, \forall a, b \in [0,1]$$

then we say that *C* is a compensatory operator.

6.6 AVERAGING OPERATORS

A typical compensatory operator is the arithmetical mean defined as

$$MEAN\,(a, b) = \frac{a+b}{2} \qquad ...(6.29)$$

Fuzzy set theory provides a host of attractive aggregation connectives for integrating membership values representing uncertain information. These connectives can be categorized into the following three classes union, intersection and compensation connectives.

Union produces a high output whenever any one of the input values representing degrees of satisfaction of different features or criteria is high.

Intersection connectives produce a high output only when all of the inputs have high values. Compensative connectives have the property that a higher degree of satisfaction of one of the criteria can compensate for a lower degree of satisfaction of another criteria to a certain extent.

In the sense, union connectives provide full compensation and intersection connectives provide no compensation. In a decision process the idea of trade-offs corresponds to viewing the global evaluation of an action as lying between the worst and the best local ratings. This occurs in the presence of conflicting goals, when a compensation between the corresponding compabilities is allowed. Averaging operators realize trade-offs between objectives, by allowing a positive compensation between ratings.

6.6.1 An Averaging Operator is a Function

$$M : [0, 1] \times [0, 1] \to [0, 1] \qquad ...(6.30)$$

satisfying the following properties

- Idempotency

$$M(x, x) = x,\ \forall x \in [0, 1] \qquad ...(6.31)$$

- Commutativity

$$M(x,y) = M(y, x),\ \forall x, y \in [0, 1] \qquad ...(6.32)$$

- Extremal conditions

$$M(0, 0) = 0, M(1, 1) = 1 \qquad ...(6.33)$$

- Monotonicity

$$M(x, y) \le M(x', y'), \text{ if } x \le x' \text{ and } y \le y' \qquad ...(6.34)$$

- M is continuous.

Averaging operators represent a wide class of aggregation operators. We prove that whatever is the particular definition of an averaging operator, M, the global evaluation of an action will lie between the *worst* and the *best* local ratings:

Lemma 6.5: If M is an averaging operator then

$$\min\,\{x, y\} \le M(x, y) \le \max\,\{x, y\},\ \forall x, y \in [0, 1].$$

Proof: From idempotency and monotonicity of M it follows that

$$\min\{x, y\} = M(\min\{x, y\}, \min\{x, y\}) \le M(x, y)$$

$$\text{and } M\{x, y\} \le M(\max\{x, y\}, \max\{x, y\}) = \max\{x, y\}$$

Which ends the proof.

Averaging operators have the following interesting properties:

Property 1. A strictly increasing averaging operator cannot be associative.

Property 2. The only associative averaging operators are defined by

$$M(x, y, \alpha) = \text{med}(a, y, \alpha) = \begin{cases} y & \text{if} \quad x \le y \le \alpha \\ \alpha & \text{if} \quad x \le \alpha \le y \\ x & \text{if} \quad \alpha \le x \le y \end{cases} \qquad ...(6.35)$$

where $\alpha \in (0, 1)$

An important family of averaging operators is formed by quasi-arithmetic means

$$M(a_1, ...\alpha_n) = f^{-1}\left(\frac{1}{n}\sum_{i=1}^{n} f(a_1)\right)$$

This family has been characterized by Kolmogorov as being the class of all decomposable continuous averaging operators. For example, the quasi-arithmetic mean of a_1 and a_2 is defined by

$$M(a_1, a_2) = f^{-1}\left(\frac{f(a_1) + f(a_2)}{2}\right) \qquad ...(6.36)$$

The next table shows the most often used mean operators.

Table 6.1 Mean operators

Name	$M(x, y)$
Harmonic mean	$2xy/(x + y)$
Geometric mean	$\sqrt{xy}$
Arithmetic mean	$(x + y)/2$
Dual of geometric mean	$1 - \sqrt{(1-x)(1-y)}$
Dual of harmonic mean	$(x + y - 2xy)/(2 - x - y)$
Median	med (x, y, α), $\alpha \in (0, 1)$
Generalized p-mean	$\left((x^p + y^p)/2\right)^{1/p}$, $p \ge 1$

6.6.2 Ordered Weighted Averaging

The process of information aggregation appears in many applications related to the development of intelligent systems. One sees aggregation in neural networks, fuzzy logic controllers, vision systems, expert systems and multi-criteria decision aids. In 1988 Yager introduced a new aggregation technique based on the ordered weighted averaging (OWA) operators.

An OWA operator of dimension n is mapping F: $\Re^n \rightarrow \Re$, that has an associated weighting vector $W = (w_1, w_2,...w_n)^T$ such as $w_i \in [0, 1]$, $1 \leq i \leq n$ and $w_1 + w_2 +...+ w_n = 1$.

Furthermore

$$F(a_1, a_2,...a_n) = w_1b_1 + w_2b_2 +...+ w_nb_n = \sum_{j=1}^{n} w_jb_j \qquad ...(6.37)$$

where b_j is the j-th largest element of the bag $(a_1,..., a_n)$.

Example 6.3: Assume $W = (0.4, 0.3, 0.2, 0.1)^T$, then

$$F(0.7, 1, 0.2, 0.6) = 0.4 \times 1 + 0.3 \times 0.7 + 0.2 \times 0.6 + 0.1 \times 0.2 = 0.75.$$

A fundamental aspect of this operator is the re-ordering step, in particular an aggregate a_i is not associated with a particular weight w_i but rather a weight is associated with a particular ordered position of aggregate. When we view the OWA weights as a column vector we shall find it convenient to refer to the weights with the low indices as weights at the top and those with the higher indices with weights at the bottom.

It is noted that different OWA operators are distinguished by their weighting function. In 1988 Yager pointed out three important special cases of OWA aggregations:

- F^*: In this case $W = W^* = (1, 0,...,0)^T$ and

$$F^*(a_1, a_2, ..., a_n) = \max \{a_1, a_2, ..., a_n\} \qquad ...(6.38)$$

- F_*: In this case $W = W_* = (1, 0, ..., 0)^T$ and

$$F_*(a_1, a_2, ..., a_n) = \min \{a_1, a_2, ..., a_n\} \qquad ...(6.39)$$

- F_A: In this case $W = W_A = (1/n ..., 1/n)^T$ and

$$F_A(a_1, a_2,..., a_n) = \frac{a_1 + ... + a_n}{n} \qquad ...(6.40)$$

A number of important properties can be associated with the OWA operators. We shall now discuss some of these. For any OWA operator F holds

$$F_*(a_1, a_2, ..., a_n) \leq F(a_1, a_2, ..., a_n) \leq F^*(a_1, a_2, ..., a_n) \qquad ...(6.41)$$

Thus the upper an lower star OWA operator are its boundaries. From the above it becomes clear that for any F

$$\min \{a_1, a_2, ..., a_n\} \leq F(a_1, a_2, ..., a_n) \leq \max \{a_1, a_2, ..., a_n\} \qquad ...(6.42)$$

The OWA operator can be seen to be *commutative*. Let $(a_1, a_2, ..., a_n)$ be a bag of aggregates and let $\{d_1, ..., d_n\}$ be any *permutation* of the a_i. Then for any OWA operator

$$F(a_1, a_2, ..., a_n) = F(d_1, d_2, ..., d_n) \quad ...(6.43)$$

A third characteristic associated with these operators is *monotonicity*. Assume a_i and c_i are a collection of aggregates, $i = 1, ..., n$ such that for each i, $a_i \geq c_i$. Then

$$F(a_1, a_2, ..., a_n) \geq (c_1, c_2, ..., c_n) \quad ...(6.44)$$

where F is some fixed weight OWA operator.

Another characteristic associated with these operators is *idempotency*. If $a_i = a$ for all i then for any OWA operator

$$F(a_1, ..., a_n) = a. \quad ...(6.45)$$

From the above we can see the OWA operators have the basic properties associated with an *averaging operator*.

Example 6. 4: A window type OWA operator takes the average of the m arguments around the center. For this class of operators we have

$$w_i = \begin{cases} 0 & \text{if} \quad i < k \\ \dfrac{1}{m} & \text{if} \quad k \leq i \leq k + m \\ 0 & \text{if} \quad i \geq k + m \end{cases}$$

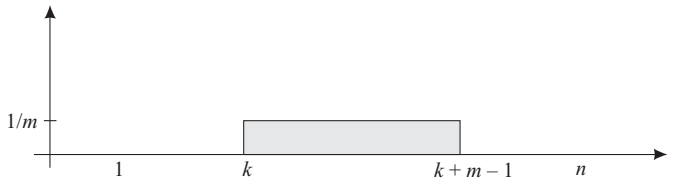

Fig. 6.1 Window type OWA operator.

In order to classify OWA operators in regard to their location between *and* and *or*, a measure of *orness*, associated with any vector W is introduce by Yager as follows

$$\text{orness } (W) = \frac{1}{n-1} \sum_{i=1}^{n} (n - i) w_i$$

It is easy to see that for any W the orness(W) is always in the unit interval. Furthermore, note that the nearer W is to an *or*, the closer its measure is to none; while the nearer it is to an *and*, the closer is to zero.

Lemma 6.6: Let us consider the vectors

$$W^* = (1, 0, ..., 0)^T, \quad W_* = (1, 0, ..., 0)^T \text{ and } \quad W_A = (1/n, ..., 1/n)^T.$$

Then it can easily be shown that

$$\text{orness}\,(W^*) = 1, \quad \text{orness}\,(W_*) = 0 \text{ and } \quad \text{orness}\,(W_A) = 0.5.$$

A measure of andness is defined as

$$\text{andness}\,(W) = 1 - \text{orness}\,(W).$$

Generally, an OWA opeartor with much of non-zero weights near the top will be an *orlike* operator, that is,

$$\text{orness}\,(W) \geq 0.5$$

and when much of the weights are non-zero near the bottom, the OWA operator will be *andlike*, that is,

$$\text{andness}\,(W) \geq 0.5.$$

Example 6.5: Let $W = (0.8, 0.2, 0.0)^T$. Then

$$\text{orness}\,(W) = \frac{1}{3}(2 \times 0.8 + 0.2) = 0.6 \text{ and}$$

$$\text{andness}\,(W) = 1 - \text{orness}\,(W) = 1 - 0.6 = 0.4$$

This means that the OWA operator, defined by

$$F(a_1, a_2, a_3) = 0.8b_1 + 0.2b_2 + 0.0b_3$$

where b_j is the j-th largest element of the bag (a_1, a_2, a_3) is an or like aggregation.

The following theorem shows that as we move weight up the vector we increase the *orness*, while moving weight down causes us to decrease orness(W).

Theorem 6.1: (Yager, 1993) Assume W and W' are two n-dimensional OWA vectors such that

$$W = (w_1, \ldots, w_n)^T$$

and

$$W' = (w_1, \ldots, w_j + \varepsilon, \ldots w_k - \varepsilon, \ldots w_n)^T \qquad \ldots(6.46)$$

where $\varepsilon > 0$, $j < k$. Then orness (W') > orness (W)

Proof: From the definition of the measure of orness we get

$$\text{orness}\,(W') = \frac{1}{n-1}\sum_i (n-i)\, w_1' = \frac{1}{n-1}\sum_i (n-i)w_i + (n-j)\varepsilon - (n-k)\varepsilon$$

$$\text{orness}\,(W') = \text{orness}\,(W) + \frac{1}{n-1}\varepsilon(k-j)$$

since $k > j$, orness (W') > orness (W).

6.7 MEASURE OF DISPERSION OR ENTROPY OF AN OWA VECTOR

In 1988 Yager defined the measure of dispersion (or entropy) of an OWA vector by

$$\text{disp}\ (W) = \sum_i w_i \ln w_i \qquad ...(6.47)$$

We can see when using the OWA operator as an averaging operator Disp (W) measures the degree to which we use all the aggregates equally.

We can see when using the OWA operator as an averaging operator $Disp(W)$ measures the degree to which we use all the aggregates equally.

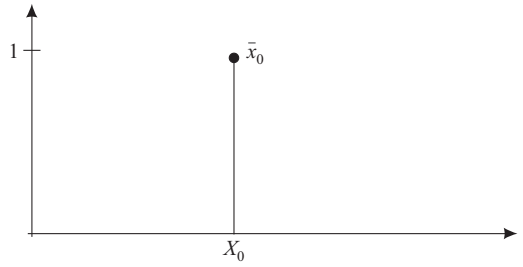

Fig. 6.2 Fuzzy singleton.

Suppose now that the fact of the GMP is given by a fuzzy singleton. Then the process of computation of the membership function of the consequence becomes very simple.

For example, if we use Mamdani's implication operator in the GMP then

Rule 1:	if x is A_1 then z is C_1
Fact:	x is $\bar{x}_0$
consequence:	z is C

where the membership function of the consequence C is computed as

$$C(w) = \sup_u \min \{ \bar{x}_0 (u), (A_1 \to C_1) (u, w)\} = \sup_u \min \{ \bar{x}_0 (u), \min \{A_1(u), C_1(w)\}\} \qquad ...(6.48)$$

for all w.

Observing that $\bar{x}_0 (u) = 0$, $\forall u \neq x_0$, the supremum turns into a simple minimum

$$C(w) = \min \{ \bar{x}_0 (x_0) \wedge A_1 (x_0) \wedge C_1(w)\} = \min \{1 \wedge A_1(x_0) \wedge C_1(w)\} = \min \{A_1(x_0), C_1(w)\} \qquad ...(6.49)$$

for all w.

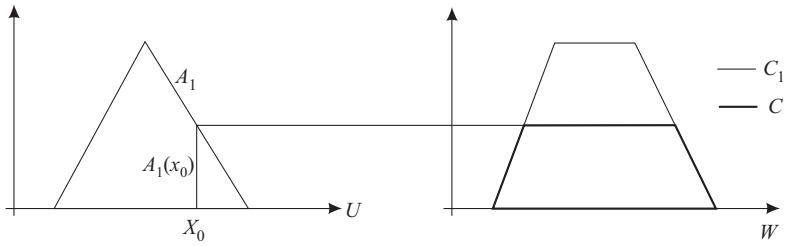

Fig. 6. 3: Inference with Mamdani's implication operator.

If we use Godel implication operator in the GMP, then

$$C(w) = \sup_u \min \{ \bar{x}_0 (u), (A_1 \to C_1) (u, w)\} = A_1(x_0) \to C_1(w) \quad ...(6.50)$$

for all w.

So,

$$C(w) = \begin{cases} 1 & \text{if} \quad A_1(x_0) \le C_1(w) \\ C_1(w) & \text{otherwise} \end{cases} \quad ...(6.51)$$

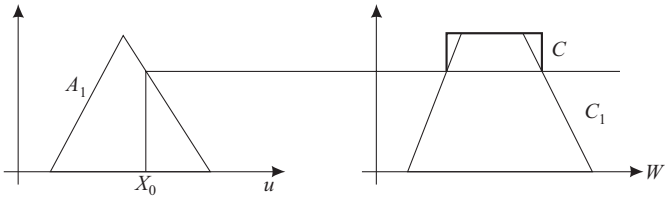

Fig. 6.4 Inference with Godel implication operator.

Rule 1:	if x is A_1 then z is C_1
Fact	x is $\bar{x}_0$
Consequence:	z is C

where the membership function of the consequence C is computed as

$$C(w) = \sup_u \min \{ \bar{x}_0 (u), (A_1 \to C_1) (u, w)\} = A_1(x_0) \to C_1(w) \quad ...(6.52)$$

for all w.

Consider a block of fuzzy IF-THEN rules

R_1 :	if x is A_1 then z is C_1
also	
R_2:	if x is A_2 then z is C_2
also	
.....	
also	
R_n :	if x is A_n then z is C_n
fact:	x is $\bar{x}_0$
Consequence:	z is C

The i-th fuzzy rule from this rule-base

$$R_i : \text{if } x \text{ is } A_i \text{ then } z \text{ is } C_i \qquad ...(6.53)$$

is implemented by a *fuzzy implication* R_i and is defined as

$$R_i(u, w) = (A_i \to C_i)(u, w) = A_i(u) \to C_i(w) \qquad ...(6.54)$$

for $i = 1,..., n$.

Find C from the input x_0 and from the rule base

$$R = \{R_1, ..., R_n\} \qquad ...(6.55)$$

Interpretation of

- sentence connective "also"
- implication operator "then"
- compositional operator "o"

We first compose $\bar{x}_0$ with each R_i producing intermediate result

$$C'_1 = \bar{x}_0 \text{ o } R_i \qquad ...(6.56)$$

for $i = 1,..., n$.

C'_1 is called the output of the i-th rule

$$C'_1(w) = A_i(x_0) \to C_i(w) \qquad ...(6.57)$$

for each w.

Then combine the C'_1 component wise into C' by some aggregation operator:

$$C = \bigcup_{i=1}^{n} C'_1 = \bar{x}_0 \text{ o } R_1 \cup ... \cup \bar{x}_0 \text{ o } R_n \qquad ...(6.58)$$

$$C(w) = A_1(x_0) \to C_1(w) \vee ... \vee$$
$$A_n(x_0) \to C_n(w)$$

So, the inference process is the following

- input to the system is x_0
- fuzzified input is $\bar{x}_0$
- firing strength of the i-th rule is $A_i(x_0)$
- the i-th individual rule output is

$$C'_1(w) := A_1(x_0) \rightarrow C_1(w) \quad ...(6.59)$$

- overall system output (action) is

$$C = C'_1 \cup ... \cup C'_n \quad ...(6.60)$$

Overall system output = union of the individual rule outputs.

6.8 MAMDANI SYSTEM

For the Mamdani (Fig. 6.5) system

$$(a \rightarrow b = a \wedge b) \quad ...(6.61)$$

- input to the system is x_0
- fuzzified input is $\bar{x}_0$
- firing strength of the i-th rule is $A_i(x_0)$
- the i-th individual rule output is

$$C'_1(w) = A_i(x_0) \wedge C_i(w) \quad ...(6.62)$$

- overall system output (action) is

$$C(w) = \bigvee_{i=1}^{n} A_i(x_0) \wedge C_i(w) \quad ...(6.63)$$

6.9 LARSEN SYSTEM

For the Larsen (Fig. 6.6) system

$$(a \rightarrow b = ab) \quad ...(6.64)$$

- input to the system is x_0
- fuzzified input is $\bar{x}_0$
- firing strength of the i-th rule is $A_i(x_0)$
- the i-th individual rule output is

$$C'_1(w) = A_i(x_0)\, C_i(w) \quad ...(6.65)$$

- overall system output (action) is

$$C(w) = \bigvee_{i=1}^{n} A_i(x_0)\, C_i(w) \quad ...(6.66)$$

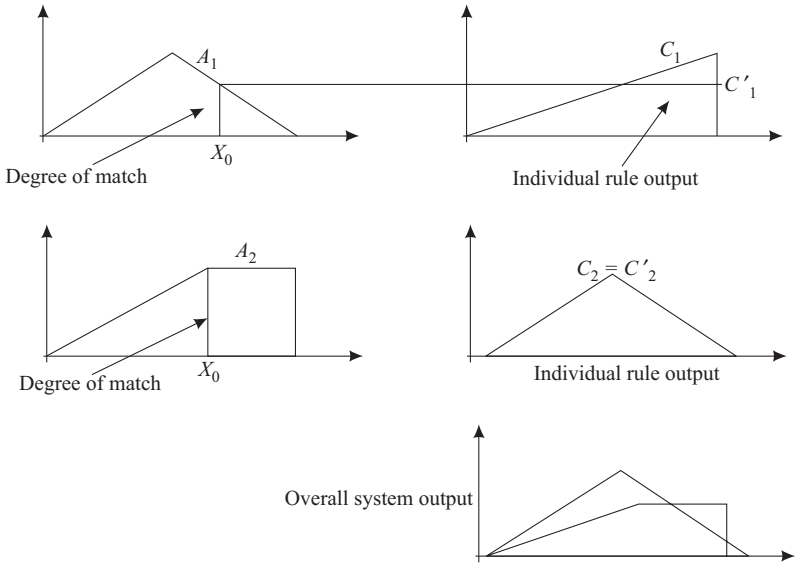

Fig. 6.5 Illustration of Mamdani system.

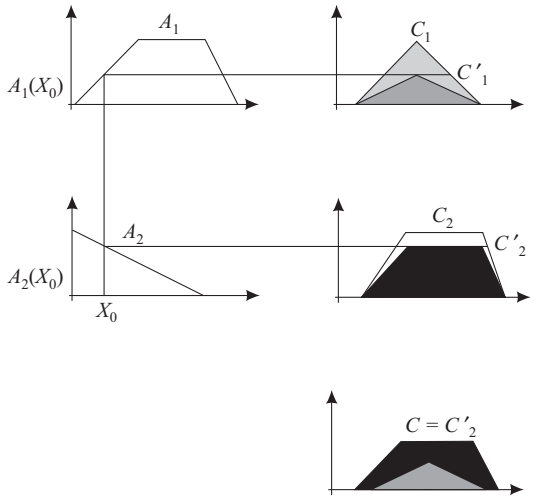

Fig. 6.6 Illustration of Larsen system.

6.10 DEFUZZIFICATION

The output of the inference process so far is a fuzzy set, specifying a possibility distribution of the (control) action. In the on-line control, a non-fuzzy (crisp) control action is usually required. Consequently, one must defuzzify the fuzzy control action (output) inferred from the fuzzy reasoning algorithm, namely:

$$z_0 = \text{defuzzifier } (C) \qquad ...(6.67)$$

where z_0 is the crisp action and *defuzzifier* is the defuzzification operator.

Defuzzification is a process to select a representative element from the fuzzy output C inferred from the fuzzy control algorithm.

QUESTION BANK

1. What is *t*-norm?
2. What are the properties to be satisfied by a *t*-norm?
3. What are the various basic *t*-norms?
4. What is *t*-conorm?
5. What are the properties to be satisfied by a *t*-conorm?
6. What are the various basic *t*-conorms?
7. Let T be a *t*-norm. Prove the following statement

$$T_W(x, y) \le T(x, y) \le \min\{x, y\}, \ \forall x, y \in [0, 1]$$

8. Let S be a *t*-conorm. Prove the following statement:

   $\max\{a, b\} \le (S(a, b) \le \text{STRONG}(a, b)), \ \forall a, b \in [0, 1]$.
9. What is *t*-norm based intersection? Explain with an example.
10. What is *t*-conorm based union? Explain with an example.
11. What are the averaging operators?
12. What are the important properties of averaging operators?
13. Explain order weighted averaging with an example.
14. Explain the Measure of dispersion.
15. What is entropy of an ordered weighted averaging (OWA) vector?
16. Explain the inference with Mamdani's implication operator.
17. Explain the inference with Godel's implication operator.
18. Explain Mamdani rule-based system.
19. Explain Larsen rule-based system.
20. What is defuzzification?

REFERENCES

1. B. Schwartz and A. Sklar, Statistical metric spaces, *Pacific Journal of Mathematics*, Vol. 10, pp. 313-334, 1960.
2. B. Schwartz and A. Sklar, Associative functions and statistical triangle inequalities, *Publication Mathematics*, *Debrecen*, Vol. 8, pp. 169-186, 1961.
3. B. Schwartz and A. Sklar, Associative functions and abstract semigroups, *Publication Mathematics*, *Debrecen*, Vol. 10, pp. 69-81, 1963.

4. E. Czogala and W. Pedrycz, Fuzzy rule generation for fuzzy control, *Cybernetics and Systems*, Vol. 13, No. 3, pp. 275-29358, 1982.
5. R.R. Yagar, Measures of fuzziness based on *t*-norms, *Stochastica*, Vol. 6, No. 1, pp. 207-229, 1982.
6. R.R. Yagar, Strong truth and rules of inference in fuzzy logic and approximate reasoning, *Cybernetics and Systems*, Vol. 16, No. 1, pp. 23-63, 1985.
7. J.A. Bernard, Use of rule-based system for process control, *IEEE Control Systems Magazine*, Vol. 8, No. 5, pp. 3-13, 1988.
8. V. Novak and W. Pedrucz, Fuzzy sets and *t*-norms in the light of fuzzy logic, *International Journal of Man-machine Studies*, Vol. 29, No. 1, pp. 113-127, 1988.
9. M.H. Lim and T. Takefuji, Implementing fuzzy rule-based systems on silicon chips, *IEEE Expert*, Vol. 5, No. 1, pp. 31-45, 1990.
10. X.T. Peng, Generating rules for fuzzy logic controllers by functions, *Fuzzy Sets and Systems*, Vol. 36, No. 1, pp. 83-89, 1990.
11. D.P. Filev and R.R. Yagar, A generalized defuzzification method via bad distributions, *International Journal of Intelligent Systems*, Vol. 6, No. 7, pp. 687-687, 1991.
12. J.C. Fodor, A remark on constructing *t*-norms, *Fuzzy Sets and Systems*, Vol. 41, No. 2, pp. 195-199, 1991.
13. M.M. Gupta and J. Qi, Theory of *t*-norms and fuzzy inference methods, *Fuzzy Sets and Systems*, Vol. 40, No. 3, pp. 431-450, 1991.
14. A. Nafarich and J.M. Keller, A fuzzy logic rule-based automatic target recognition, *International Journal of Intelligent Systems*, Vol. 6, No. 3, pp. 295-312, 1991.
15. R.R. Yagar, A general approach to rule aggregation in fuzzy logic control, *Applied Intellignece*, Vol. 2, No. 4, pp. 335-351, 1992.
16. L.X. Wang, and J.M. Mendel, Generating fuzzy rules by learning through examples, *IEEE Transactions on Systems*, *Man and Cybernetics*, Vol. 22, No. 6, pp. 1414-1427, 1992.
17. F. Bouslama and A Ichikawa, Fuzzy control rules and their natural control laws, *Fuzzy Sets and Systems*, Vol. 48, No. 1, pp. 65-86, 1992.
18. J.J. Bukley, A general theory of uncertainty based on *t*-conorms, *Fuzzy Sets and Systems*, Vol. 48, No. 3, pp. 289-296, 1992.
19. D. Dubois and H. Prade, Gradual inference rules in approximate reasoning, *Information Sciences*, Vol. 61, No. 1, pp. 103-122, 1992.
20. R. Fuller and H.J. Zimmerman, On computation of the compositional rule of inference under triangular norms, *Fuzzy Sets and Systems*, Vol. 51, No. 3, pp. 267-275, 1992.
21. D.L. Hudson, M.E. Coben and M.F. Anderson, Approximate reasoning with IF-THEN-UNLESS rule in a medical expert system, *International Journal of Intelligent Systems*, Vol. 7, No. 1, pp. 71-79, 1992.
22. F.C.H. Rhee and R. Krishanpuram, Fuzzy rule generation methods for high-level computer vision, *Fuzzy Sets and Systems*, Vol. 60, No. 3, pp. 245-258, 1993.
23. B. Cao, Input-output mathematical model with *t*-fuzzy sets, *Fuzzy Sets and Systems*, Vol. 59, No. 1, pp. 15-23, 1993.

24. P. Doherry, P. Driankov and H. Hellendoom, Fuzzy IF-THEN-UNLESS rules and their implementation, *International Journal of Uncertainity*, *Fuzziness and Knowledge-based Systems*, Vol. 1, No. 2, pp. 167-182, 1993.
25. S. Dutta and P.P. Bonissone, Integrating case and rule-based reasoning, *International Journal of Approximate Reasoning*, Vol. 8, No. 3, pp. 163-204, 1993.
26. T. Sudkamp, Similarity, interpolation and fuzzy rule construction, *Fuzzy Sets and Systems*, Vol. 58, No. 1, pp. 73-86, 1993.
27. Y.Tian and I.B. Turksen, Combination of rules or their consequences in fuzzy expert systems, *Fuzzy Sets and Systems*, Vol. 58, No. 1, pp. 3-40, 1993.
28. E. Uchino, T. Yamakawa, T. Miki and S. Nakamura, Fuzzy rule-based simple interpolation algorithm for discrete signal, *Fuzzy Sets and Systems*, Vol. 59, No. 3, pp. 259-270, 1993.
29. T. Arnould and S. Tano, A rule-based method to calculate exactly the widest solutions sets of a max-min fuzzy relations inequality, *Fuzzy Sets and Systems*, Vol. 64, No. 1, pp. 39-58, 1994.
30. V. Cross and T. Sudkamp, Patterns of fuzzy-rule based interference, *International Journal of Approximate Reasoning*, Vol. 11, No. 3, pp. 235-255, 1994.
31. C. Ebert, Rule-based fuzzy classification for software quality control, *Fuzzy Sets and Systems*, Vol. 63, No. 3, pp. 349-358, 1994.
32. J. Kacprzyk, On measuring the specificity of IF-THEN rules, *International Journal of Approximate Reasoning*, Vol. 11, No. 1, pp. 29-53, 1994.
33. W. Pedrycz, Why triangular membership functions? *Fuzzy Sets and Systems*, Vol. 64, No. 1, pp. 21-30, 1994.

CHAPTER 7

Fuzzy Reasoning Schemes

7.1 INTRODUCTION

This chapter focuses different inference mechanisms in fuzzy rule-based systems with examples. The inference engine of a fuzzy expert system operates on a series of production rules and makes fuzzy inferences.

There exist two approaches to evaluating relevant production rules. The first is data-driven and is exemplified by the generalized modus ponens. In this case, available data are supplied to the expert system, which then uses them to evaluate relevant production rules and draw all possible conclusions. An alternative method of evaluation is goal-driven; it is exemplified by the generalized modus tollens form of logical inference. Here, the expert system searches for data specified in the *IF* clauses of production rules that will lead to the objective; these data are found either in the knowledge base, in the *THEN* clauses of other production rules, or by querying the user.

Since the data-driven method proceeds from *IF* clauses to *THEN* clauses in the chain through the production rules, it is commonly called forward chaining. Similarly, since the goal-driven method proceeds backward from *THEN* clauses to the *IF* clauses, in its search for the required data, it is commonly called backward chaining. Backward chaining has the advantage of speed, since only the rules leading to the objective need to be evaluated.

7.2 FUZZY RULE-BASE SYSTEM

R_1 : if x is A_1 and y is B_1 then z is C_1

R_2 : if x is A_2 and y is B_2 then z is C_2

. . . . . .

R_n : if x is A_n and y is B_n then z is C_n

x is $\bar{x}_0$ and y is $\bar{y}_0$

---

z is C

The i-th fuzzy rule from this rule-base

$$R_i : \text{if } x \text{ is } A_i \text{ and } y \text{ is } B_i \text{ then } z \text{ is } C_i$$

is implemented by a fuzzy relation R_i and is defined as

$$R_i(u, v, w) = (A_i \times B_i \to C_i)(u, w) = [A_i(u) \wedge B_i(v)] \to C_i(w) \quad ...(7.1)$$

for $i = 1, ..., n$.

Find C from the input x_0 and from the rule base

$$R = \{R_1, ..., R_n\} \quad ...(7.2)$$

Interpretation of

- logical connective "and"
- sentence connective "also"
- implication operator "then"
- compositional operator "o"

We first compose $\bar{x}_0 \times \bar{y}_0$ with each R_i producing intermediate result

$$C_1' = \bar{x}_0 \times \bar{y}_0 \text{ o } R_i \quad ...(7.3)$$

for $i = 1, ..., n$. Here C_1' is called the output of the i-th rule

$$C_1'(w) = [A_i(x_0) \wedge B_i(y_0)] \to C_i(w) \quad ...(7.4)$$

for each w.

Then combine the C_1' component wise into C' by some aggregation operator:

$$C = \bigcup_{i=1}^{n} C_1' = \bar{x}_0 \times \bar{y}_0 \text{ o } R_1 \cup ... \cup \bar{x}_0 \times \bar{y}_0 \text{ o } R_n$$

$$C(w) = A_i(x_0) \times B_i(y_0) \to C_1(w) \vee ... \vee \quad ...(7.5)$$
$$A_n(x_0) \times B_n(y_0) \to C_n(w)$$

- input to the system is (x_0, y_0)
- fuzzified input is $(\bar{x}_0, \bar{y}_0)$
- firing strength of the i-th rule is $A_i(x_0) \wedge B_i(y_0)$
- the i-th individual rule output is $C_1'(w) := A_1(x_0) \wedge B_1(x_0) \to C_1(w)$
- overall system output is $C = C_i' \cup ... \cup C_n'$.

overall system output = union of the individual rule outputs.

7.3 INFERENCE MECHANISMS IN FUZZY RULE-BASE SYSTEMS

We present five well-known inference mechanisms in fuzzy rule-based systems.

For simplicity we assume that we have two fuzzy *IF-THEN* rules of the form

R_1 :	if x is A_1 and y is B_1 then z is C_1
also	
R_2 :	if x is A_2 and y is B_2 then z is C_2
fact:	x is $\bar{x}_0$ and y is $\bar{y}_0$
Consequence:	z is C

7.3.1 Mamdani Inference Mechanism

The fuzzy implication is modelled by Mamdani's minimum operator and the sentence connective also is interpreted as oring the propositions and defined by max operator.

The firing levels of the rules, denoted by α_i, $i = 1, 2$, are computed by

$$\alpha_1 = A_1(x_0) \wedge B_1(y_0),\ \alpha_2 = A_2(x_0) \wedge B_2(y_0) \quad ...(7.6)$$

The individual rule outputs are obtained by

$$C_1'(w) = (\alpha_1 \wedge C_1(w)),\ C_2'(w) = (\alpha_2 \wedge C_2(w)) \quad ...(7.7)$$

Then the overall system output is computed by oring the individual rule outputs

$$C(w) = C_1'(w) \vee C_2'(w) = (\alpha_1 \wedge C_1(w)) \vee (\alpha_2 \wedge C_2(w)) \quad ...(7.8)$$

Finally, to obtain a deterministic control action, we employ any defuzzification strategy.

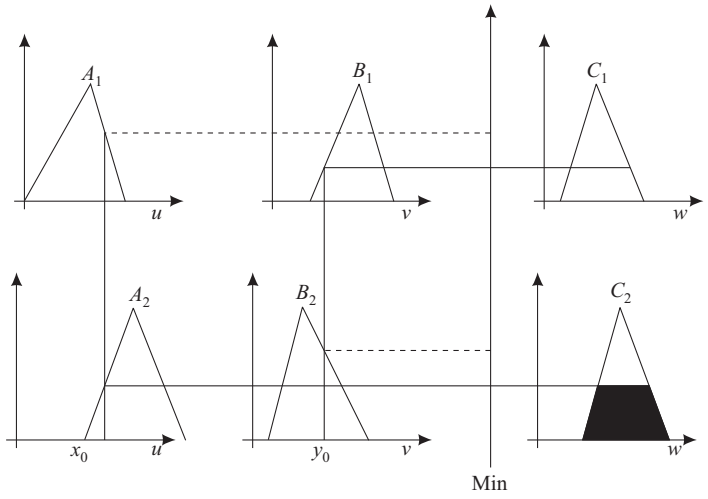

Fig. 7.1 Inference with Mamdani's implication operator.

7.3.2 Tsukamoto Inference Mechanism

All linguistic terms are supposed to have monotonic membership functions. The firing levels of the rules, denoted by α_i, $i = 1, 2$, are computed by

$$\alpha_1 = A_1(x_0) \wedge B_1(y_0),\ \alpha_2 = A_2(x_0) \wedge B_2(y_0) \quad ...(7.9)$$

In this mode of reasoning the individual crisp control actions z_1 and z_2 are computed from the equations

$$\alpha_1 = C_1(z_1),\ \alpha_2 = C_2(z_2) \qquad ...(7.10)$$

and the overall crisp control action is expressed as

$$z_0 = \frac{\alpha_1 z_1 + \alpha_2 z_2}{\alpha_1 + \alpha_2} \qquad ...(7.11)$$

i.e. z_0 is computed by the discrete Center of-Gravity method.

If we have n rules in our rule-base then the crisp control action is computed as

$$z_0 = \frac{\sum_{i=1}^{n} \alpha_i z_i}{\sum_{i=1}^{n} \alpha_i} \qquad ...(7.12)$$

where α_i is the firing level and z_i is the (crisp) output of the i-th rule, $i = 1,..., n$

Example 7.1: We illustrate Tsukamoto's reasoning method by the following simple example

R_1 :	if x is A_1 and y is B_1 then z is C_1
also	
R_2 :	if x is A_2 and y is B_2 then z is C_2
fact:	x is $\bar{x}_0$ and y is $\bar{y}_0$
Consequence:	z is C

Then according to the figure we see that

$$A_1(x_0) = 0.7,\ B_1(y_0) = 0.3$$

Therefore, the firing level of the first rule is

$$\alpha_1 = \min\{A_1(x_0), B_1(y_0)\} = \min\{0.7, 0.3\} = 0.3$$

and from

$$A_2(x_0) = 0.6.\ B_2(y_0) = 0.8$$

It follows that the firing level of the second rule is

$$\alpha_2 = \min\{A_2(x_0), B_2(y_0)\} = \min\{0.6, 0.8\} = 0.6$$

The individual rule outputs $z_1 = 8$ and $z_2 = 4$ are derived from the equations

$$C_1(z_1) = 0.3,\ C_2(z_2) = 0.6$$

and the crisp control action is

$$z_0 = (8 \times 0.3 + 4 \times 0.6)/(0.3 + 0.6) = 6$$

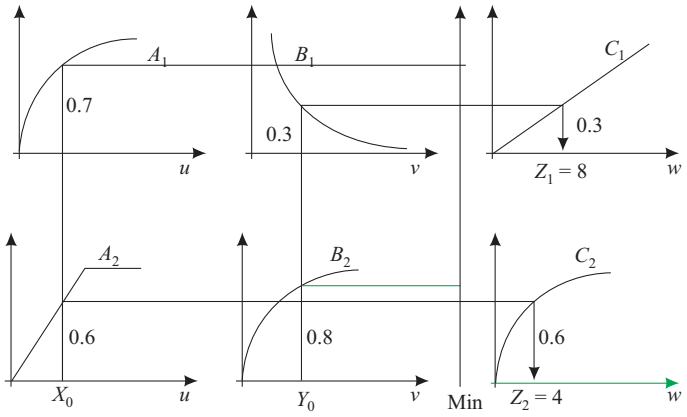

Fig. 7.2 Tsukamoto's inference mechanism.

7.3.3 Sugeno Inference Mechanism

Sugeno and Takagi use the following architecture

R_1 :	if x is A_1 and y is B_1 then $z_1 = a_1x + b_1y$
also	
R_2 :	if x is A_2 and y is B_2 then $z_2 = a_2x + b_2y$
fact:	x is $\bar{x}_0$ and y is $\bar{y}_0$
Consequence:	z_0

The firing levels of the rules are computed by

$$\alpha_1 = A_1(x_0) \wedge B_1(y_0),\ \alpha_2 = A_2(x_0) \wedge B_2(y_0) \qquad ...(7.13)$$

then the individual rule outputs are derived from the relationships

$$z_1^* = a_1x_0 + b_1y_0,\ z_2^* = a_2x_0 + b_2y_0 \qquad ...(7.14)$$

and the crisp control action is expressed as

$$z_0 = \frac{\alpha_1 z_1^* + \alpha_2 z_2^*}{\alpha_1 + \alpha_2} \qquad ...(7.15)$$

If we have n rules in our rule-base then the crisp control action is computed as

$$z_0 = \frac{\sum_{i=1}^{n} \alpha_1 z_i^*}{\sum_{i=1}^{n} \alpha_i} \qquad ...(7.16)$$

where α_i denotes the firing level of the i-th rule, $i = 1, ..., n$.

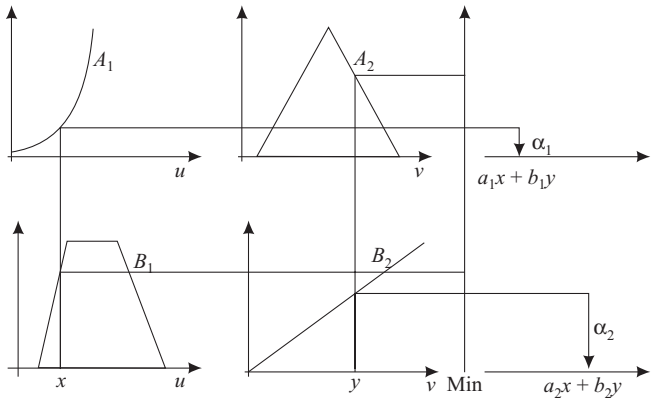

Fig. 7.3 Sugeno's inference mechanism.

Example 7.2: We illustrate Sugeno's reasoning method by the following simple example

R_1 :	if x is *BIG* and y *is SMALL* then $z_1 = x + y$
also	
R_2 :	if x is *MEDIUM* and y *is BIG* then $z_2 = 2x - y$
fact :	x is 3 and y is 2
Consequence:	z_0

Then according to the figure we see that

$$\mu_{BIG}(x_0) = \mu_{BIG}(3) = 0.8$$

$$\mu_{SMALL}(y_0) = \mu_{SMALL}(2) = 0.2$$

Therefore, the firing level of the first rule is

$$\alpha_1 = \min\{\mu_{BIG}(x_0), \mu_{SMALL}(y_0)\} = \min\{0.8, 0.2\} = 0.2$$

and from

$$\mu_{MEDIUM}(x_0) = \mu_{MEDIUM}(3) = 0.6, \mu_{BIG}(y_0) = \mu_{BIG}(2) = 0.9$$

It follows that the firing level of the second rule is

$$\alpha_2 = \min\{\mu_{MEDIUM}(x_0), \mu_{BIG}(y_0)\} = \min\{0.6, 0.9\} = 0.6$$

The individual rule outputs are computed as

$$z_1^* = x_0 + y_0 = 3 + 2 = 5, \quad z_2^* = 2x_0 - y_0 = 2 \times 3 - 2 = 4$$

So the crisp control action is

$$z_0 = (5 \times 0.2 + 4 \times 0.6)/(0.2 + 0.6) = 4.25$$

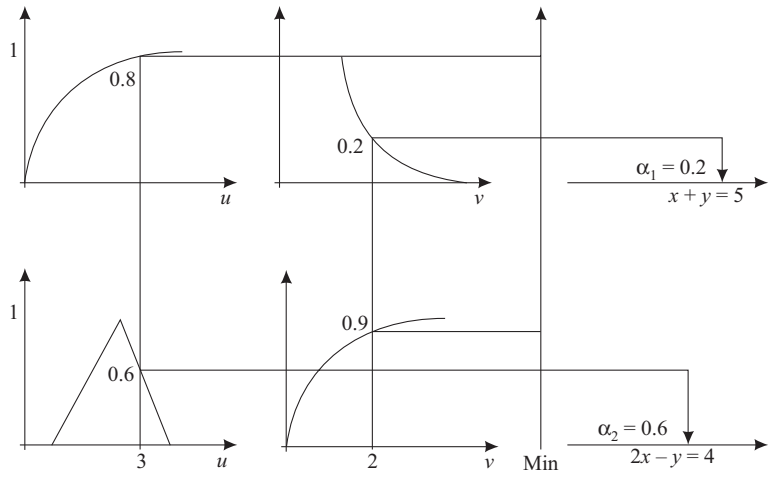

Fig. 7.4 Example of Sugeno's inference mechanism.

7.3.4 Larsen Inference Mechanism

The fuzzy implication is modeled by Larsen's product operator and the sentence connective also is interpreted as oring the propositions and defined by max operator.

Let us denote α_i the firing level of the i-th rule, $i = 1, 2$

$$\alpha_1 = A_1(x_0) \wedge B_1(y_0),\ \alpha_2 = A_2(x_0) \wedge B_2(y_0) \qquad ...(7.17)$$

Then membership function of the inferred consequence C is pointwise given by

$$C(w) = (\alpha_1 C_1(w)) \vee (\alpha_2 C_2(w)) \qquad ...(7.18)$$

To obtain a deterministic control action, we employ any defuzzification strategy.

If we have n rules in our rule-base then the consequence C is computed as

$$C(w) = \bigvee_{i=1}^{n} (\alpha_i C_i(w)) \qquad ...(7.19)$$

where α_i denotes the firing level of the i-th rule, $i = 1, ..., n$

7.3.5 Simplified Fuzzy Reasoning

R_1 :	if x is A_1 and y is B_1 then $z_1 = C_1$
also	
R_2 :	if x is A_2 and y is B_2 then $z_2 = C_2$
fact:	x is $\bar{x}_0$ and y is $\bar{y}_0$
Consequence:	z_0

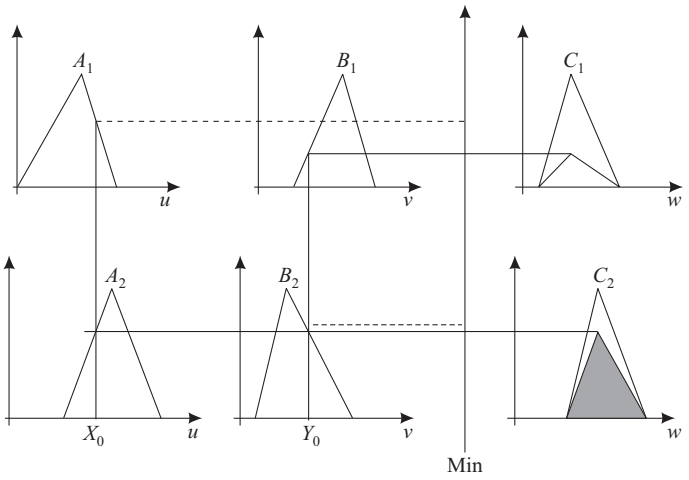

Fig. 7.5 Inference with Larsen's product operation rule.

The firing levels of the rules are computed by

$$\alpha_1 = A_1(x_0) \wedge B_1(y_0),\ \alpha_2 = A_2(x_0) \wedge B_2(y_0) \qquad ...(7.20)$$

then the individual rule outputs are c_1 and c_2, and the crisp control action is expressed as

$$z_0 = \frac{\alpha_1 c_1 + \alpha_2 c_2}{\alpha_1 + \alpha_2} \qquad ...(7.21)$$

If we have n rules in our rule-base then the crisp control action is computed as

$$z_0 = \frac{\sum_{i=1}^{n} \alpha_i C_i}{\sum_{i=1}^{n} \alpha_i} \qquad ...(7.22)$$

where α_i denotes the firing level of the i-th rule, $i = 1, ..., n$.

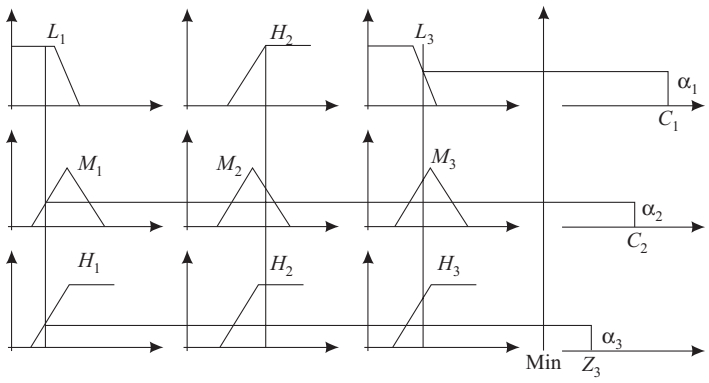

Fig. 7.6 Simplified fuzzy reasoning.

QUESTION BANK

1. What are the different approaches to evaluating relevant production rules? Explain them.
2. Explain Mamdani inference mechanism.
3. Explain Tsukamoto inference mechanism.
4. Explain Sugeno inference mechanism.
5. Explain Larsen inference mechanism.
6. Explain simplified reasoning scheme.

REFERENCES

1. L.A. Zadeh, Fuzzy logic and approximate reasoning, *Synthese*, Vol. 30, No. 1, pp. 407-428, 1975.
2. L.A. Zadeh, The concept of a linguistic variable and its application to approximate reasoning I, *Information Sciences*, Vol. 8, pp. 199-251, 1975.
3. L.A. Zadeh, The concept of a linguistic variable and its application to approximate reasoning II, *Information Sciences*, Vol. 8, pp. 301-357, 1975.
4. L.A. Zadeh, The concept of a linguistic variable and its application to approximate reasoning III, Information sciences, Vol. 9, pp. 43-80, 1975.
5. B.R. Gaines, Foundations of fuzzy reasoning, *International Journal of Man-machine Studies*, Vol. 8, No. 6, pp. 623-668, 1976.
6. E.H. Mamdani, Applications of fuzzy logic to approximate reasoning using linguistic systems, *IEEE Transactions on Systems*, *Man and Cybernetics*, Vol. 26, No. 12, pp. 1182-1191, 1977.
7. J.F. Baldwin, Fuzzy logic and reasoning, *International Journal of Man-machine Studies*, Vol. 11, No. 4, pp. 465-480, 1979.
8. E.H. Mamdani and B.R. Gaines, *Fuzzy Reasoning and Its Applications*, Academic Press, London, 1981.
9. M. Sugeno and T. Takagi, Multidi-mensional fuzzy reasoning, *Fuzzy Sets and Systems*, Vol. 9, No. 3, pp. 313-325, 1983.
10. W. Pedrycz, Applications of fuzzy relational equations for methods of reasoning in presence of fuzzy data, *Fuzzy Sets and Systems*, Vol. 16, No. 2, pp. 163-175, 1985.
11. H. Farreny and H. Prade, Default and inexact reasoning with possibility degrees, *IEEE Transactions on Systems*, *Man and Cybernetics*, Vol. 16, No. 2, pp. 270-276, 1986.
12. M.B. Gorzalczany, A method of inference in approximate reasoning based on interval-valued fuzzy sets, *Fuzzy Sets and Systems*, Vol. 21, No. 1, pp. 1-17, 1987.
13. E. Sanchez and L.A. Zadeh, *Approximate Reasoning in Intelligent Systems*, *Decision and Control*, Pergamon Press, Oxford, U.K, 1987.
14. I.B. Turksen, Approximate reasoning for production planning, *Fuzzy Sets and Systems*, Vol. 26, No. 1, pp. 23-37, 1988.
15. I.B. Turksen, Four methods of approximate reasoning with interval-valued fuzzy sets, *International Journal of Approximate Reasoning*, Vol. 3, No. 2, pp. 121-142, 1989.

16. A. Basu and A. Dutta, Reasoning with imprecise knowledge to enhance intelligent decision support, *IEEE Transactions on Systems*, man and cybernetics, Vol. 19, No. 4, pp. 756-770, 1989.
17. Z. Cao, A. Kandel and L. Li, A new model for fuzzy reasoning, *Fuzzy Sets and Systems*, Vol. 36, No. 3, pp. 311-325, 1990.
18. R. Kruse and E. Schwecke, Fuzzy reasoning in a multidimensional space of hypotheses, *International Journal of Approximate Reasoning*, Vol. 4, No. 1, pp. 47-68, 1990.
19. C.Z. Luo and Z.P. Wang, Representation of compositional relations in fuzzy reasoning, *Fuzzy Sets and Systems*, Vol. 36, No. 1, pp. 77-81, 1990.
20. D. Dubois and H. Prade, Fuzzy sets in approximate reasoning, Part I: Inference with possibility distributions, *Fuzzy Sets and Systems*, Vol. 40, No. 1, pp. 143-202, 1991.
21. S. Dutta, Approximate spatial reasoning: Integrating qualitative and quantitative constraints, *International Journal of Approximate Reasoning*, Vol. 5, No. 3, pp. 307-330, 1991.
22. Z. Pawlak, Rough sets: Theoretical aspects of reasoning about data, Kluwer, Bostan, 1991.
23. E.H. Ruspini, Approximate reasoning: past, present, future, *Information Sciences*, Vol. 57, pp. 297-317, 1991.
24. S.M. Chen, A new improved algorithm for inexact reasoning based on extended fuzzy production rules, *Cybernetics and Systems*, Vol. 23, No. 5, pp. 409-420, 1992.
25. D.L. Hudson, M.E. Coben and M.F. Anderson, Approximate reasoning with IF-THEN-UNLESS rule in a medical expert system, *International Journal of Intelligent Systems*, Vol. 7, No. 1, pp. 71-79, 1992.
26. H. Nakanishi, I.B. Turksen and M. Sugeno, A review and comparison of six reasoning methods, *Fuzzy Sets and Systems*, Vol. 57, No. 3, pp. 257-294, 1993.
27. Z. Bien and M.G. Chun, An inference network for bidirectional approximate reasoning based on an equality measure, *IEEE Transactions on Fuzzy Systems*, Vol. 2, No. 2, pp. 177-180, 1994.

CHAPTER 8

Fuzzy Logic Controllers

8.1 INTRODUCTION

Conventional controllers are derived from control theory techniques based on mathematical models of the open-loop process, called *system*, to be controlled. The purpose of the feedback controller is to guarantee a desired response of the output y. The process of keeping the output y close to the set point (reference input) y^*, despite the presence disturbances of the system parameters, and noise measurements, is called regulation. The output of the controller (which is the input of the system) is the control action u.

8.2 BASIC FEEDBACK CONTROL SYSTEM

The general form of the discrete-time control law is

$$u(k) = f(e(k), e(k-1), ..., e(k-\tau), u(k-1), ..., e(k-\tau)) \quad ...(8.1)$$

providing a control action that describes the relationship between the input and the output of the controller.

- e represents the error between the desired set point y^* and the output of the system y,
- parameter τ defines the order of the controller,
- f is in general a non-linear function.

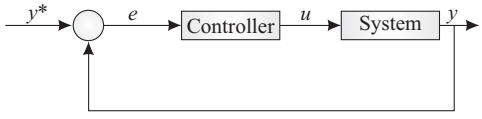

Fig. 8.1 A basic feedback control system.

8.3 FUZZY LOGIC CONTROLLER

L.A. Zadeh (1973) was introduced the idea of formulating the control algorithm by logical rules. In a fuzzy logic controller (FLC), the dynamic behaviour of a fuzzy system is characterized by a set of linguistic description rules based on expert knowledge. The expert knowledge is usually of the form

IF (a set of conditions are satisfied) **THEN** (a set of consequences can be inferred).

Since the antecedents and the consequents of these IF-THEN rules are associated with fuzzy concepts (linguistic terms), they are often called fuzzy conditional statements.

In our terminology, a fuzzy control rule is a fuzzy conditional statement in which the antecedent is a condition in its application domain and the consequent is a control action for the system under control.

Basically, fuzzy control rules provide a convenient way for expressing control policy and domain knowledge. Furthermore, several linguistic variables might be involved in the antecedents and the conclusions of these rules. When this is the case, the system will be referred to as a multi-input-multi-output (MIMO) fuzzy system.

8.3.1 Two-Input-Single-Output (TISO) Fuzzy Systems

For example, in the case of two-input-single-output fuzzy systems, fuzzy control rules have the form.

R_1 : if x is A_1 and y is B_1 then z is C_1

also

R_2 : if x is A_2 and y is B_2 then z is C_2

also

. . .

also

R_n : if x is A_n and y is B_n then z is C_n

where x and y are the process state variables, z is the control variable, A_i, B_i, and C_i are linguistic values of the linguistic variables x, y and z in the universes of discourse U, V, and W, respectively, and an implicit sentence connective also links the rules into a rule set or, equivalently, a rule-base.

8.3.2 Mamdani Type of Fuzzy Logic Control

We can represent the FLC in a form similar to the conventional control law

$$u(k) = f(e(k), e(k-1), ..., e(k-\tau), u(k-1), ..., e(k-\tau) \quad ...(8.2)$$

where the function F is described by a fuzzy rule base. However, it does not mean that the *FLC* is a kind of transfer function or difference equation.

The knowledge-based nature of FLC dictates a limited usage of the past values of the error e and control u because it is rather unreasonable to expect meaningful linguistic statements for

$$e(k-3), e(k-4), ..., e(k-\tau).$$

A typical *FLC* describes the relationship between the changes of the control

$$\Delta u(k) = u(k) - u(k-1) \qquad ...(8.3)$$

On the one hand, and the error $e(k)$ and its change

$$\Delta e(k) = e(k) - e(k-1) \qquad ...(8.4)$$

On the other hand, such control law can be formalized as

$$\Delta u(k) = F(e(k), \Delta e(k)) \qquad ...(8.5)$$

and is a manifestation of the general *FLC* expression with $\tau = 1$.

The actual output of the controller $u(k)$ is obtained from the previous value of control $u(k-1)$ that is updated by $\Delta u(k)$

$$u(k) = u(k-1) + \Delta u(k). \qquad ...(8.6)$$

This type of controller was suggested originally by *Mamdani and Assilian* in 1975 and is called the *Mamdani type FLC*. A prototypical rule-base of a simple FLC realizing the control law above is listed in the following

R_1 : if e is "positive" and Δe is "near zero" then Δu is "positive"

R_2 : if e is "negative" and Δe is "near zero" then Δu is "negative"

R_3 : if e is "near zero" and Δe is "near zero" then Δu is "near zero"

R_4 : if e is "near zero" and Δe is "positive" then Δu is "positive"

R_5 : if e is "near zero" and Δe is "negative" then Δu is "negative"

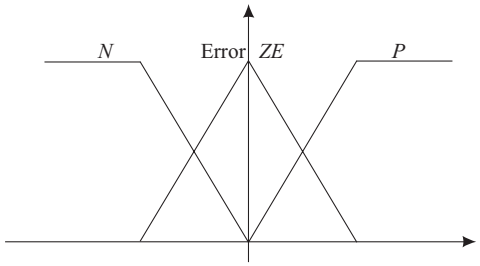

Fig. 8.2 Membership functions for the error.

So, our task is the find a crisp control action z_0 from the fuzzy rule-base and from the actual crisp inputs x_0 and y_0:

R_1 : if x is A_1 and y is B_1 then z is C_1

also

R_2 : if x is A_2 and y is B_2 then z is C_2

also

. . . .

also

R_n : if x is A_n and y is B_n then z is C_n

input x is $\bar{x}_0$ and y is $\bar{y}_0$

output z_0

Of course, the inputs of fuzzy rule-based systems should be given by fuzzy sets, and therefore, we have to fuzzify the crisp inputs. Furthermore, the output of a fuzzy system is always a fuzzy set, and therefore to get crisp value we have to defuzzify it.

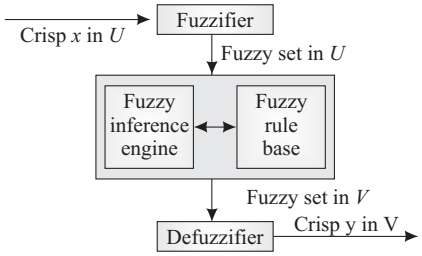

Fig. 8.3 Fuzzy logic controller.

8.3.3 Fuzzy Logic Control Systems

Fuzzy logic control systems (Figure 8.3) usually consist of four major parts:

- Fuzzification interface,
- Fuzzy rulebase,
- Fuzzy inference machine and
- Defuzzification interface.

A fuzzification operator has the effect of transforming crisp data into fuzzy sets. In most of the cases we use fuzzy singletons as fuzzifiers

$$\text{fuzzifier } (x_0) := \bar{x}_0 \qquad ...(8.7)$$

where x_0 is a crisp input value from a process.

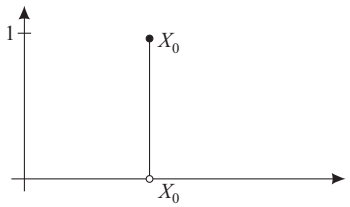

Fig. 8.4 Fuzzy singleton as fuzzifier.

Suppose now that we have two input variables x and y. A fuzzy control rule

$$R_i : \text{if } (x \text{ is } A_i \text{ and } y \text{ is } B_i \text{ then } (z \text{ is } C_i)$$

is implemented by a *fuzzy implication* R_i and is defined as

$$R(u,v,w) = [A_i(u) \text{ and } B_i(v)] \rightarrow C_i(w) \qquad ...(8.8)$$

where the logical connective *and* is implemented by the minimum operator, i.e.

$$[A_i(u) \text{ and } B_i(v)] \rightarrow C_i(w) = [A_i(u) \times B_i(v)] \rightarrow C_i(w) \qquad ...(8.9)$$

$$= \min \{[A_i(u), B_i(v)] \rightarrow C_i(w)\}$$

Of course, we can use any t-norm to model the logical connective *and*. Fuzzy control rules are combined by using the sentence connective *also*. Since each fuzzy control rule is represented by a fuzzy relation, the overall behavior of a fuzzy system is characterized by these fuzzy relations.

In other words, a fuzzy system can be characterized by a single fuzzy relation which is the combination in question involves the sentence connective *also*.

Symbolically, if we have the collection of rules

R_1 : if x is A_1 and y is B_1 then z is C_1
also
R_2 : if x is A_2 and y is B_2 then z is C_2
also
. . .
also
R_n : if x is A_n and y is B_n then z is C_n

The procedure for obtaining the fuzzy output of such a knowledge base consists from the following three steps:

- Find the firing level of each of the rules.
- Find the output of each of the rules.
- Aggregate the individual rule outputs to obtain the overall system output.

To infer the output z from the given process states x, y and fuzzy relations R_i, we apply the compositional rule of inference:

R_1 : if x is A_1 and y is B_1 then z is C_1
also
R_2 : if x is A_2 and y is B_2 then z is C_2
also
. . .
also
R_n : if x is A_n and y is B_n then z is C_n
input x is $\bar{x}_0$ and y is $\bar{y}_0$

Consequence : z is C

Where the consequence is computed by

$$\text{consequence} = \text{Agg (fact o } R_1, ..., \text{ fact o } R_n) \quad ...(8.10)$$

That is,

$$C = \text{Agg } (\bar{x}_0 \times \bar{y}_0 \text{ o } R_1, ..., \bar{x}_0 \times \bar{y}_0 \text{ o } R_n) \quad ...(8.11)$$

taking into consideration that

$$\bar{x}_0 (u) = 0, u \neq x_0 \quad ...(8.12)$$

and

$$\bar{y}_0 (v) = 0, v \neq y_0 \quad ...(8.13)$$

The computation of the membership function of C is very simple:

$$C(w) = \text{Agg } \{A_1(x_0) \times B_1(y_0) \rightarrow C_1(w), ..., A_n(x_0) \times B_n(y_0) \rightarrow C_n(w)\} \quad ...(8.14)$$

for all $w \in W$.

The procedure for obtaining the fuzzy output of such a knowledge base can be formulated as

- The firing level of the I-th rule is determined by

$$A_i(x_0) \times B_i(y_0) \quad ...(8.15)$$

- The output of the I-th rule is calculated by

$$C'_1(w) = A_i(x_0) \times B_i(y_0) \rightarrow C_i(w) \text{ for all } w \in W \quad ...(8.16)$$

- The overall system output, C, is obtained from the individual rule outputs C'_i by

$$C(w) = \text{Agg } \{C'_1, ..., C'_n\} \text{ for all } w \in W. \quad ...(8.17)$$

Example 8.1: If the sentence connective also is interpreted as oring the rules by using minimum-norm then the membership function of the consequence is computed as

$$C = (\bar{x}_0 \times \bar{y}_0 \text{ o } R_1 \cup ... \cup \bar{x}_0 \times \bar{y}_0 \text{ o } R_n)$$

That is

$$C(w) = A_1(x_0) \times B_1(y_0) \rightarrow C_1(w) \; V ... V \; A_n(x_0) \times B_n (y_0) \rightarrow C_n(w)$$

for all $w \in W$.

8.4 DEFUZZIFICATION METHODS

The output of the inference process so far is a fuzzy set, specifying a possibility distribution of control action. In the on-line control, a nonfuzzy (crisp) control action is usually required.

Consequently, one must defuzzify the fuzzy control action (output) inferred from the fuzzy control algorithm, namely:

$$z_0 = \text{defuzzifier } (C) \quad ...(8.18)$$

where z_0 is the nonfuzzy control output and *defuzzifier* is the defuzzification operator.

Defuzzification is a process to select a representative element from the fuzzy output C inferred from the fuzzy control algorithm.

The most often used defuzzification operators are:

8.4.1 Center-of-Area/Gravity

The defuzzified value of a fuzzy set C is defined as its fuzzy centroid:

$$z_0 = \frac{\int_w zC(z)dz}{\int_w c(z)dz} \qquad ...(8.19)$$

The calculation of the Center-of-Area defuzzified value is simplified if we consider finite universe of discourse W and thus discrete membership function $C(w)$

$$z_0 = \frac{\sum z_j C(z_j)dz}{\sum c(z_j)} \qquad ...(8.20)$$

8.4.2 First-of-Maxima

The defuzzified value of a fuzzy set C is its smallest maximizing element, i.e.

$$z_0 = \min \left\{ z \middle| C(z) = \max_u C(w) \right\} \qquad ...(8.21)$$

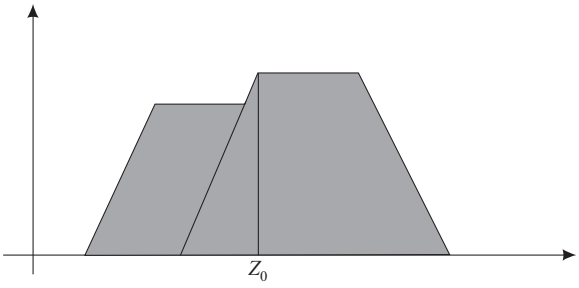

Fig. 8.5 First-of-maxima defuzzification method.

8.4.3 Middle-of-Maxima

The defuzzified value of a discrete fuzzy set C is defined as a mean of all values of the universe of discourse, having maximal membership grades

$$z_0 = \frac{1}{N}\sum_{j=1}^{n} z_j \qquad ...(8.22)$$

where $\{z_1, ..., z_N\}$ is the set of elements of the universe W which attain the maximum value of C.

If C is not discrete then defuzzified value of a fuzzy set C is defined as

$$z_0 = \frac{\int_G z dz}{\int_G dz} \qquad ...(8.23)$$

where G denotes the set of maximizing element of C.

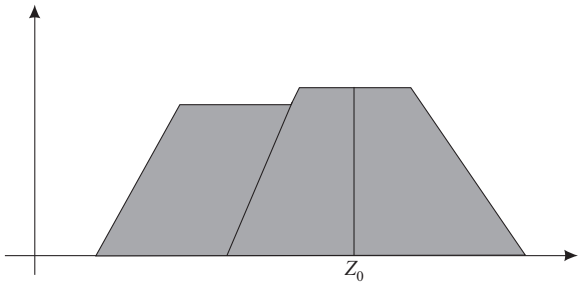

Fig. 8.6 Middle-of-maxima defuzzification method.

8.4.4 Max-Criterion

This method chooses an arbitrary value, from the set of maximizing elements of C, i.e.

$$z_0 \in \left\{ z \middle| C(z) = \max_w C(w) \right\} \qquad ...(8.24)$$

8.4.5 Height Defuzzification

The elements of the universe of discourse W that have membership grades lower than a certain level α are completely discounted and the defuzzified value z_0 is calculated by the application of the Center-of-Area method on those elements of W that have membership grades not less than α:

$$z_0 = \frac{\int_{[C]^\alpha} zC(z)dz}{\int_{[C]^\alpha} c(z)dz} \qquad ...(8.25)$$

where $[C]^\alpha$ denotes the α-level set of C as usually.

Example 8.2: Consider a fuzzy controller steering a car in a way to avoid obstacles. If an obstacle occurs right ahead, the plausible control action depicted in Figure could be interpreted as "**turn right or left"**

Both Center-of-Area and Middle-of-Maxima defuzzification methods result in a control action "driveahead straightforward" which causes an accident.

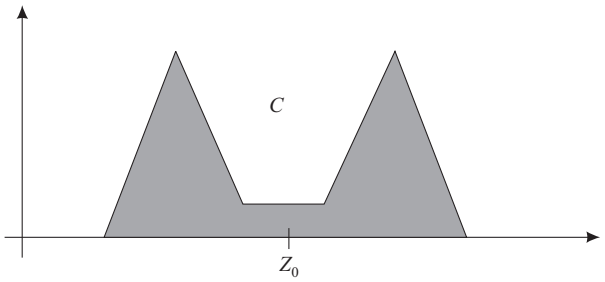

Fig. 8.7 Undesired result by Center-of-Area and Middle-of-Maxima defuzzification methods.

A suitable defuzzification method would have to choose between different control actions (choose one of two triangles in the Figure) and then transform the fuzzy set into a crisp value.

8.5 EFFECTIVITY OF FUZZY LOGIC CONTROL SYSTEMS

Using the Stone-Weierstrass theorem, Wang (1992) showed that fuzzy logic control systems of the form

R_i: if x is A_i and y is B_i then z is C_i, $i = 1, \ldots, n$

with

- Gaussian membership functions

$$A_i(u) = \exp\left[-\frac{1}{2}\left(\frac{u-\alpha_{i1}}{\beta_{i1}}\right)^2\right]$$

$$B_i(u) = \exp\left[-\frac{1}{2}\left(\frac{v-\alpha_{i2}}{\beta_{i2}}\right)^2\right] \qquad ...(8.26)$$

$$C_i(w) = \exp\left[-\frac{1}{2}\left(\frac{w-\alpha_{i3}}{\beta_{i3}}\right)^2\right]$$

- Singleton fuzzifier

$$\text{fuzzifier } (x): = \bar{x}\text{ , fuzzifier } (y): = \bar{y} \qquad ...(8.27)$$

- Product fuzzy conjunction

$$[A_i(u) \text{ and } B_i(v)] = A_i(u)\, B_i(v) \qquad ...(8.28)$$

- Product fuzzy implication (Larsen implication)

$$[A_i(u) \text{ and } B_i(\text{v})] \rightarrow C_i(w) = A_i(u)\, B_i(v)\, C_i(w) \qquad (8.29)$$

- Centroid defuzzification method

$$z = \frac{\sum_{i=1}^{n} \alpha_{i3} A_i(x) B_i(y)}{\sum_{i=1}^{n} A_i(x) B_i(y)} \quad \text{... (8.30)}$$

where α_{i3} is the center of C_i.

are universal approximators, i.e. they can approximate any continuous function on a compact set to arbitrary accuracy.

Namely, he proved the following theorem

Theorem 8.1 *For a given real-valued continuous function g on the compact set U and arbitrary* $\varepsilon > 0$*, there exists a fuzzy logic control system with output function f such that*

$$\sup_{x \in U} \|g(x) - f(x)\| \leq \varepsilon \quad \text{...(8.31)}$$

Castro in 1995 showed that Mamdani's fuzzy logic controllers

$$R_i: \text{ if } x \text{ is } A_i \text{ and } y \text{ is } B_i \text{ then } z \text{ is } C_i, \; i = 1, \ldots, n$$

with

- Symmetric triangular membership functions

$$A_i(u) = \exp \begin{cases} 1 - |a_i - u|/\alpha_i & \text{if } |a_i - u| \leq \alpha_i \\ 0 & \text{otherwise} \end{cases}$$

$$B_i(v) = \exp \begin{cases} 1 - |b_i - v|/\beta_i & \text{if } |b_i - v| \leq \beta_i \\ 0 & \text{otherwise} \end{cases} \quad \text{...(8.32)}$$

$$C_i(w) = \exp \begin{cases} 1 - |c_i - w|/\gamma_i & \text{if } |c_i - w| \leq \gamma_i \\ 0 & \text{otherwise} \end{cases}$$

- Singleton fuzzier

$$\text{fuzzifier } (x_0) := \bar{x}_0 \quad \text{...(8.33)}$$

- Minimum norm fuzzy conjunction

$$[A_i(u) \text{ and } B_i(v)] = \min \{A_i(u) B_i(v)\} \quad \text{... (8.34)}$$

- Minimum norm fuzzy implication

$$[A_i(u) \text{ and } B_i(v)] \rightarrow C_i(W) = \min \{A_i(u), B_i(v), C_i(W)\} \quad \text{...(8.35)}$$

- Maximum *t*-conorm rule aggregation

$$\text{Agg } (R_1, R_2, \ldots, R_n) = \max \{R_1, R_2, \ldots, R_n\} \quad \text{...(8.36)}$$

- Centroid defuzzification method

$$z = \frac{\sum_{i=1}^{n} c_i \min \{A_i(x) B_i(y)\}}{\sum_{i=1}^{n} \min \{A_i(x) B_i(y)\}} \quad ...(8.37)$$

where c_i is the center of C_i are also universal approximators.

QUESTION BANK

1. What is fuzzy logic controller?
2. Explain two-input-single-output fuzzy system.
3. Explain Mamdani type of fuzzy logic controller.
4. What are the various parts of fuzzy logic control system? Explain them.
5. What are the various defuzification methods? Explain them.
6. What is the effectivity of fuzzy logic control systems?

REFERENCES

1. L.A. Zadeh, a rationale for fuzzy control, Journal of dynamical systems, *Measurement and Control*, Vol. 94, No. 1, pp. 3-4, 1971.
2. E.H. Mamdani and S. Assilian, An experiment in linguistic synthesis with a fuzzy logic controller, *International Journal of Man*, *Machine Studies*, Vol. 7, No. 1, pp. 1-13, 1975.
3. E.H. Mamdani, Advances in the linguistic synthesis of fuzzy controllers, *International Journal of Machine Studies*, Vol. 8, No. 6, pp. 669-678, 1976.
4. P.J. King and E.H. Mamdani, The application of fuzzy control systems to industrial process, *Automatica*, Vol. 13, No. 3, pp. 235-242, 1977.
5. W.J.M. Kickert and E.H. Mamdani, Analysis of a fuzzy logic controller, Fuzzy sets and systems, Vol. 1, No. 1, pp. 29-44, 1978.
6. M. Brase and D.A. Rutherford, Selection of parameters for a fuzzy logic controller, *Fuzzy Sets and Systems*, Vol. 2, No. 3, pp. 185-199, 1979.
7. C.C. Lee, Selection of parameters for a fuzzy logic controller, *Fuzzy Sets and Systems*, Vol. 2, No. 3, pp. 185-199, 1979.
8. E. Czogala and W. Pedrycz, Control problems in fuzzy systems, *Fuzzy Sets and Systems*, Vol. 7, No. 3, pp. 257-274, 1982.
9. E. Czogala and W. Pedrycz, Fuzzy rule generation for fuzzy control, *Cybernetics and Systems*, Vol. 13, No. 3, pp. 275-293, 1982.

10. K.S. Ray and D. Dutta Majumdar, Application of circle criteria for stability analysis of linear SISO and MIMO systems associated with fuzzy logic controller, *IEEE Transactions of on Systems*, *Man and Cybernetics*, Vol. 14, No. 2, pp. 345-349, 1984.
11. M. Sugeno, An introductory survey of fuzzy control, *Infromation Sciences*, Vol. 36, No. 1, pp. 59-83, 1985.
12. T. Takagi and M. Sugeno, Fuzzy identification of systems and its applications to modeling and control, *IEEE Transactions on Systems*, *Man and Cybernetics*, Vol. 15, No. 1, pp. 116-132, 1985.
13. M.M. Gupta, J.B. Kiszks and G.M. Trojan, Multivariable structure of fuzzy control systems, *IEEE Transactions on Systems*, *Man and Cybernetics*, Vol. 16, No. 5, pp. 638-656, 1986.
14. J.A. Bernard, Use of rule-based system for process control, *IEEE Control Systems Magazine*, Vol. 8, No. 5, pp. 3-13, 1988.
15. B.P. Graham and R.B. Newell, Fuzzy identification and control of a liquid level rig, *Fuzzy Sets and Systems*, Vol. 26, No. 3, pp. 255-273, 1988.
16. J.J. Buckley, Fuzzy v/s non-fuzzy controllers, *Control and Cybernetics*, Vol. 18, No. 2, pp. 127-130, 1989.
17. X.T. Peng, Generating rules for fuzzy logic controllers by functions, *Fuzzy Sets and Systems*, Vol. 36, No. 1, pp. 83-89, 1990.
18. J.F. Bladwin and N.C.F. Guild, Modeling controllers using fuzzy relations, *Kybernets*, Vol. 9, No. 3, pp. 223-229, 1991.
19. J.J. Buckley, Theory of the fuzzy controller: an introduction, *Fuzzy Sets and Systems*, Vol. 51, No. 3, pp. 249-258, 1992.
20. K. Tanaka and M. Sugeno, Stability analysis and design of fuzzy control systems, *Fuzzy Sets and Systems*, Vol. 45, No. 2, pp. 135-156, 1992.
21. G.M. Abdelnour, C.H. Chang, F.H. Huang and J.Y. Cheung, Design of a fuzzy controller using input and output mapping factors, *IEEE Transactions on Systems*, *Man and Cybernetics*, Vol. 21, No. 5, pp. 925-960, 1991.
22. F. Boullama and A. Ichikawa, Fuzzy control rules and their natural control laws, *Fuzzy Sets and Systems*, Vol. 48, No .1, pp. 65-86, 1992.
23. A. Kandel , L.H. Li and Z.Q. Cao, Fuzzy inference and its applicability to control systems, *Fuzzy Sets and Systems*, Vol. 48, No. 1, pp. 99-111, 1992.
24. R.R. Yager, A general approach to rule aggregation in fuzzy logic control, *Applied Intelligence*, Vol. 2, No. 4, pp. 335-351, 1992.
25. C. Wong, C. Chou and D. Mon, Studies on the output of fuzzy controller with multiple inputs, *Fuzzy Sets and Systems*, Vol. 57, No. 2, pp. 149-158, 1993.
26. R. Ragot and M. Lamotte, Fuzzy logic control, *International Journal of Systems Sciences*, Vol. 24, No. 10, pp. 1825-1848, 1993.
27. B. Chung and J. Oh, Control of dynamic systems using fuzzy learning algorithm, *Fuzzy Sets and Systems*, Vol. 59, No. 1, pp. 1-14, 1993.
28. J.Q. Chen and L.J. Chen, Study on stability of fuzzy closed-loop control systems, *Fuzzy Sets and Systems*, Vol. 57, No. 2, pp. 159-168, 1993.

29. N. Kiupel and P.M. Frank, Fuzzy control of steam turbines, *Journal of Systems Science*, Vol. 24, No. 10, pp.1905-1914, 1993.
30. D.P. Filev and R.R. Yagar, Three models of fuzzy logic controllers, *Cybernetics and Systems*, Vol. 24, No. 2, pp. 91-114, 1993.
31. W. Pedrycz, Fuzzy controllers: Principles and architectures, *Asia-Pacific Engineering Journal*, Vol. 3, No. 1, pp. 1-32, 1993.
32. J.Y. Han and V. Mc Murray, Two-layer multiple-variable Fuzzy logic controller, *IEEE Transactions of Systems*, *Man and Cybernetics*, Vol. 23, No. 1, pp. 277-285, 1993.
33. C.V. Altrock, H.O. Arend, B. Krause, C. Steffess and E.B. Rommler, Adaptive fuzzy control applied to home heating system, *Fuzzy Sets and Systems*, Vol. 61, No. 1, pp. 29-36, 1994.
34. R.R. Yager, and D.P. Filev, *Essentials of Fuzzy Modeling and Control*, John Wiley, New York, 1994.
35. A.J. Bugarin, S. Barro and R. Ruiz, Fuzzy control architectures, *Journal of Intelligent and Fuzzy Systems*, Vol.2, No.2, pp.125-146, 1994.

CHAPTER 9

Fuzzy Logic Applications

9.1 WHY USE FUZZY LOGIC?

Here is a list of general observations about fuzzy logic:

1. Fuzzy logic is conceptually easy to understand.

   The mathematical concepts behind fuzzy reasoning are very simple. What makes fuzzy nice is the "naturalness" of its approach and not its far-reaching complexity.

2. Fuzzy logic is flexible.

   With any given system, it's easy to massage it or layer more functionality on top of it without starting again from scratch.

3. Fuzzy logic is tolerant of imprecise data.

   Everything is imprecise if you look closely enough, but more than that, most things are imprecise even on careful inspection. Fuzzy reasoning builds this understanding into the process rather than tacking it onto the end.

4. Fuzzy logic can model nonlinear functions of arbitrary complexity.

   You can create a fuzzy system to match any set of input-output data. This process is made particularly easy by adaptive techniques like ANFIS (Adaptive Neuro-Fuzzy Inference Systems), which are available in the Fuzzy Logic Toolbox.

5. Fuzzy logic can be built on top of the experience of experts.

   In direct contrast to neural networks, which take training data and generate opaque, impenetrable models, fuzzy logic lets you rely on the experience of people who already understand your system.

6. Fuzzy logic can be blended with conventional control techniques.

   Fuzzy systems don't necessarily replace conventional control methods. In many cases fuzzy systems augment themand simplify their implementation.

7. Fuzzy logic is based on natural language.

   The basis for fuzzy logic is the basis for human communication. This observation underpins many of the other statements about fuzzy logic.

The last statement is perhaps the most important one and deserves more discussion. Natural language, that which is used by ordinary people on a daily basis, has been shaped by thousands of years of human history to be convenient and efficient. Sentences written in ordinary language represent a triumph of efficient communication. We are generally unaware of this because ordinary language is, of course, something we use every day. Since fuzzy logic is built.

9.2 APPLICATIONS OF FUZZY LOGIC

Fuzzy logic deals with uncertainty in engineering by attaching degrees of certainty to the answer to a logical question. Why should this be useful? The answer is commercial and practical. Commercially, fuzzy logic has been used with great success to control machines and consumer products. In the right application fuzzy logic systems are simple to design, and can be understood and implemented by non-specialists in control theory.

In most cases someone with a intermediate technical background can design a fuzzy logic controller. The control system will not be optimal but it can be acceptable. Control engineers also use it in applications where the on-board computing is very limited and adequate control is enough. Fuzzy logic is not the answer to all technical problems, but for control problems where simplicity and speed of implementation is important then fuzzy logic is a strong candidate. A cross section of applications that have successfully used fuzzy control includes:

1. Environmental
    - Air Conditioners
    - Humidifiers
2. Domestic Goods
    - Washing Machines/Dryers
    - Vacuum Cleaners
    - Toasters
    - Microwave Ovens
    - Refrigerators
3. Consumer Electronics
    - Television
    - Photocopiers
    - Still and Video Cameras – Auto-focus, Exposure and Anti-shake
    - Hi-Fi Systems
4. Automotive Systems
    - Vehicle Climate Control
    - Automatic Gearboxes
    - Four-wheel Steering
    - Seat/Mirror Control Systems

9.3 WHEN NOT TO USE FUZZY LOGIC?

Fuzzy logic is not a cure-all. When should you not use fuzzy logic? Fuzzy logic is a convenient way to map an input space to an output space. If you find it is not convenient, try something else. If a simpler solution already exists, use it. Fuzzy logic is the codification of common sense-use common sense when you implement it and you will probably make the right decision. Many controllers, for example, do a fine job without using fuzzy logic. However, if you take the time to become familiar with fuzzy logic, you will see it can be a very powerful tool for dealing quickly and efficiently with imprecision and non-linearity.

9.4 FUZZY LOGIC MODEL FOR PREVENTION OF ROAD ACCIDENTS

Traffic accidents are rare and random. However, many people died or injured because of traffic accidents all over the world. When statistics are investigated India is the most dangerous country in terms of number of traffic accidents among Asian countries. Many reasons can contribute these results, which are mainly driver fault, lack of infrastructure, environment, literacy, weather conditions etc. Cost of traffic accident is roughly 3% of gross national product. However, agree that this rate is higher in India since many traffic accidents are not recorded, for example single vehicle accidents or some accidents without injury or fatality.

In this study, using fuzzy logic method, which has increasing usage area in Intelligent Transportation Systems (*ITS)*, a model was developed which would obtain to prevent the vehicle pursuit distance automatically. Using velocity of vehicle and pursuit distance that can be measured with a sensor on vehicle a model has been established to brake pedal (slowing down) by fuzzy logic.

9.4.1 Traffic Accidents And Traffic Safety

The general goal of traffic safety policy is to eliminate the number of deaths and casualties in traffic. This goal forms the background for the present traffic safety program. The program is partly based on the assumption that high speed contributes to accidents. Many researchers support the idea of a positive correlation between speed and traffic accidents. One way to reduce the number of accidents is to reduce average speeds. Speed reduction can be accomplished by police surveillance, but also through physical obstacles on the roads. Obstacles such as flower pots, road humps, small circulation points and elevated pedestrian crossings are frequently found in many residential areas around India. However, physical measures are not always appreciated by drivers. These obstacles can cause damages to cars, they can cause difficulties for emergency vehicles, and in winter these obstacles can reduce access for snow clearing vehicles. An alternative to these physical measures is different applications of Intelligent Transportation Systems (*ITS*). The major objectives with *ITS* are to achieve traffic efficiency, by for instance redirecting traffic, and to increase safety for drivers, pedestrians, cyclists and other traffic groups.

One important aspect when planning and implementing traffic safety programs is therefore drivers' acceptance of different safety measures aimed at speed reduction. Another aspect is whether the individual's acceptance, when there is a certain degree of freedom of choice, might also be reflected in a higher acceptance of other measures, and whether acceptance of safety measures is also reflected in their perception of road traffic, and might reduce dangerous behaviour in traffic.

9.4.2 Fuzzy Logic Approach

The basic elements of each fuzzy logic system are, as shown in Figure 9.1, rules, fuzzifier, inference engine, and defuzzifier. Input data are most often crisp values. The task of the fuzzifier is to map crisp numbers into fuzzy sets (cases are also encountered where inputs are fuzzy variables described by fuzzy membership functions). Models based on fuzzy logic consist of "If-Then" rules. A typical "If-Then" rule would be:

If the ratio between the flow intensity and capacity of an arterial road is SMALL

Then vehicle speed in the flow is BIG

The fact following "If" is called a premise or hypothesis or antecedent. Based on this fact we can infer another fact that is called a conclusion or consequent (the fact following "Then"). A set of a large number of rules of the type:

If premise

Then conclusion is called a fuzzy rule base.

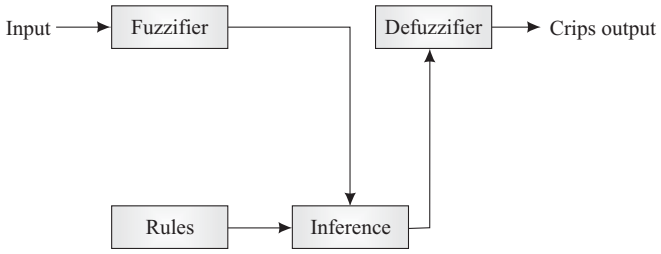

Fig. 9.1 Basic elements of a fuzzy logic.

In fuzzy rule-based systems, the rule base is formed with the assistance of human experts; recently, numerical data has been used as well as through a combination of numerical data-human experts. An interesting case appears when a combination of numerical information obtained from measurements and linguistic information obtained from human experts is used to form the fuzzy rule base. In this case, rules are extracted from numerical data in the first step. In the next step this fuzzy rule base can (but need not) be supplemented with the rules collected from human experts. The inference engine of the fuzzy logic maps fuzzy sets onto fuzzy sets. A large number of different inferential procedures are found in the literature. In most papers and practical engineering applications, minimum inference or product inference is used. During defuzzification, one value is chosen for the output variable. The literature also contains a large number of different defuzzification procedures. The final value chosen is most often either the value corresponding to the highest grade of membership or the coordinate of the center of gravity.

9.4.3 Application

In the study, a model was established which estimates brake rate using fuzzy logic. The general structure of the model is shown in Fig. 9.2.

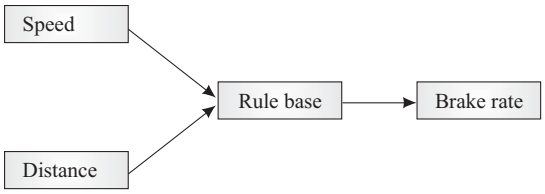

Fig. 9.2 General structure of fuzzy logic model.

9.4.4 Membership Functions

In the established model, different membership functions were formed for speed, distance and brake rate. Membership functions are given in Figures 9.3, 9.4, and 9.5. For maximum allowable car speed (in motorways) in India, speed scale selected as 0-120 km/h on its membership function. Because of the fact that current distance sensors perceive approximately 100-150 m distance, distance membership function is used 0-150 m scale. Brake rate membership function is used 0-100 scale for expressing percent type.

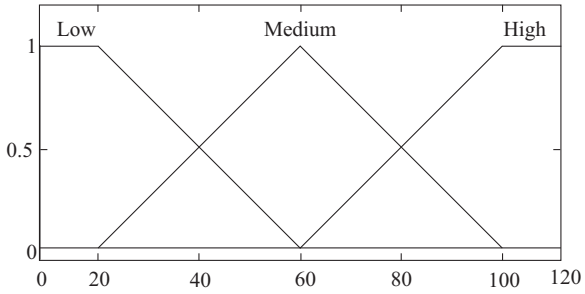

Fig. 9.3 Membership function of speed.

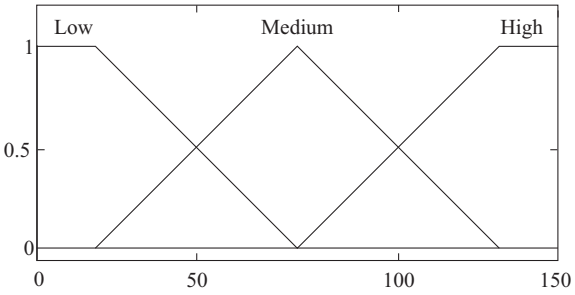

Fig. 9.4 Membership function of distance.

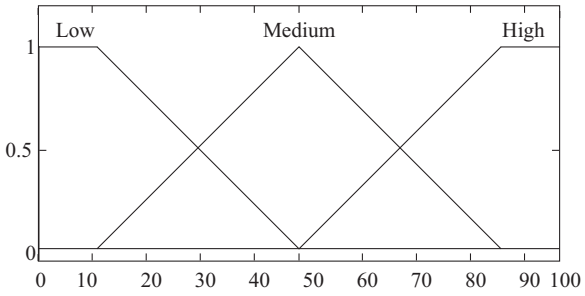

Fig. 9.5 Membership function of brake rate.

9.4.5 Rule Base

We need a rule base to run the fuzzy model. Fuzzy Allocation Map (rules) of the model was constituted for membership functions whose figures are given on Table-9.1. It is important that the rules were not completely written for all probability. Figure 6 shows that the relationship between inputs, speed and distance, and brake rate.

Table 9.1: Fuzzy allocation map of the model

Speed	*Distance*	*Brake rate*
LOW	LOW	LOW
LOW	MEDIUM	LOW
LOW	HIGH	MEDIUM
MEDIUM	LOW	MEDIUM
MEDIUM	MEDIUM	LOW
MEDIUM	HIGH	LOW
HIGH	LOW	HIGH
HIGH	MEDIUM	MEDIUM
HIGH	HIGH	LOW

9.4.6 Output

Fuzzy logic is also an estimation algorithm. For this model, various alternatives are able to cross-examine using the developed model. Fig. 9.6 is an example for such the case.

9.4.7 Conclusions

Many people die or injure because of traffic accidents in India. Many reasons can contribute these results for example mainly driver fault, lack of infrastructure, environment, weather conditions etc. In this study, a model was established for estimation of brake rate using fuzzy logic approach. Car brake rate is estimated using the developed model from speed and distance data. So, it can be said that this fuzzy logic approach can be effectively used for reduce to traffic accident rate. This model can be adapted to vehicles.

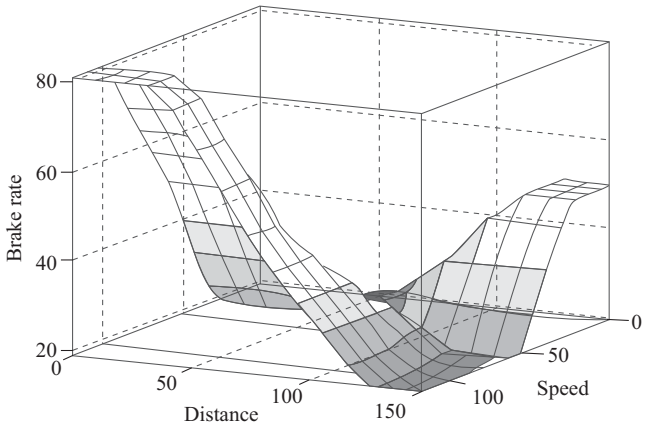

Fig. 9.6 Relationship between inputs and brake rate.

9.5 FUZZY LOGIC MODEL TO CONTROL ROOM TEMPERATURE

Although the behaviour of complex or nonlinear systems is difficult or impossible to describe using numerical models, quantitative observations are often required to make quantitative control decisions. These decisions could be the determination of a flow rate for a chemical process or a drug dosage in medical practice. The form of the control model also determines the appropriate level of precision in the result obtained. Numerical models provide high precision, but the complexity or non-linearity of a process may make a numerical model unfeasible. In these cases, linguistic models provide an alternative. Here the process is described in common language.

The linguistic model is built from a set of ***if-then*** rules, which describe the control model. Although Zadeh was attempting to model human activities, Mamdani showed that fuzzy logic could be used to develop operational automatic control systems.

9.5.1 The Mechanics of Fuzzy Logic

The mechanics of fuzzy mathematics involve the manipulation of fuzzy variables through a set of linguistic equations, which can take the form of ***if–then*** rules. Much of the fuzzy literature uses set theory notation, which obscures the ease of the formulation of a fuzzy controller. Although the controllers are simple to construct, the proof of stability and other validations remain important topics. The outline of fuzzy operations will be shown here through the design of a familiar room thermostat.

A fuzzy variable is one of the parameters of a fuzzy model, which can take one or more fuzzy values, each represented by a fuzzy set and a word descriptor. The room temperature is the variable shown in Fig. 9.7. Three fuzzy sets: ‘hot’, ‘cold’ and ‘comfortable’ have been defined by membership distributions over a range of actual temperatures.

The power of a fuzzy model is the overlap between the fuzzy values. A single temperature value at an instant in time can be a member of both of the overlapping sets. In conventional set theory, an object (in this case a temperature value) is either a member of a set or it is not a member. This implies a crisp

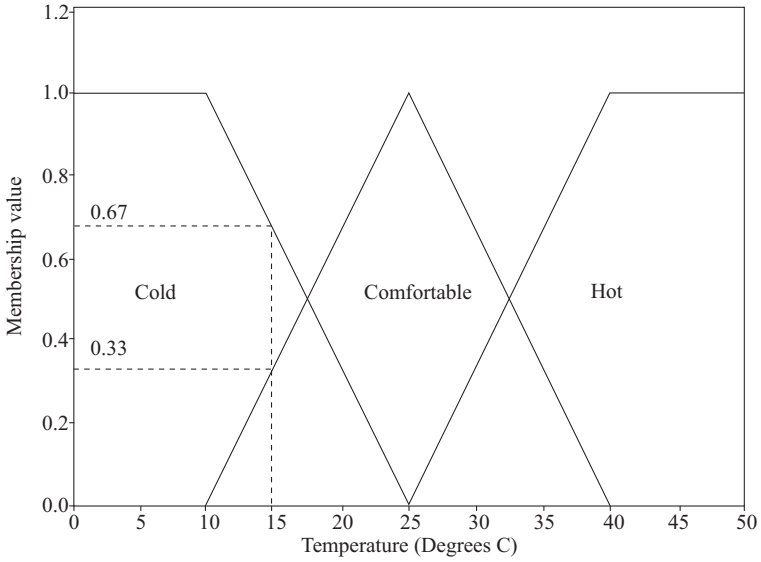

Fig. 9.7 Room temperature.

boundary between the sets. In fuzzy logic, the boundaries between sets are blurred. In the overlap region, an object can be a partial member of each of the overlapping sets. The blurred set boundaries give fuzzy logic its name. By admitting multiple possibilities in the model, the linguistic imprecision is taken into account.

The membership functions defining the three fuzzy sets shown in Fig. 9.7 are triangular. There are no constraints on the specification of the form of the membership distribution. The Gaussian form from statistics has been used, but the triangular form is commonly chosen, as its computation is simple. The number of values and the range of actual values covered by each one are also arbitrary. Finer resolution is possible with additional sets, but the computation cost increases.

Guidance for these choices is provided by Zadeh's ***Principle of Incompatibility:*** *As the complexity of a system increases, our ability to make precise and yet significant statements about its behaviour diminishes until a threshold is reached beyond which precision and significance (or relevance) become almost mutually exclusive characteristics.*

The operation of a fuzzy controller proceeds in three steps. The first is fuzzification, where measurements are converted into memberships in the fuzzy sets. The second step is the application of the linguistic model, usually in the form of ***if-then*** rules. Finally the resulting fuzzy output is converted back into physical values through a defuzzfication process.

9.5.2 Fuzzification

For a single measured value, the fuzzification process is simple, as shown in Fig. 9.7. The membership functions are used to calculate the memberships in all of the fuzzy sets. Thus, a temperature of 15°C becomes three fuzzy values, 0.66 'cold', 0.33 'comfortable' and 0.00 'hot'.

A series of measurements are collected in the form of a histogram and use this as the fuzzy input as shown in Fig. 9.8. The fuzzy inference is extended to include the uncertainty due to measurement error as well as the vagueness in the linguistic descriptions. In Fig. 9.8 the measurement data histogram is normalized so that its peak is a membership value of 1.0 and it can be used as a fuzzy set. The membership of the histogram in 'cold' is given by: max {min [$\mu_{cold}(T)$, $\mu_{histogram}(T)$]} where the maximum and minimum operations are taken using the membership values at each point T over the temperature range of the two distributions.

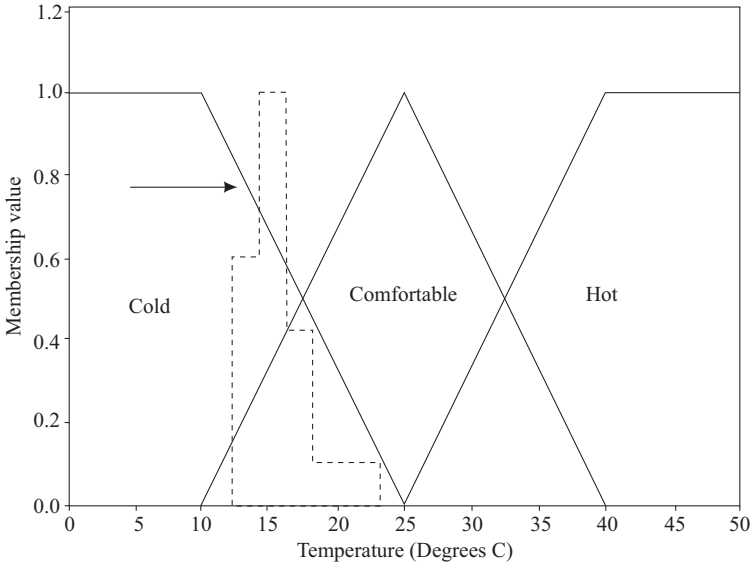

Fig. 9.8 Fuzzification with measurement noise.

The minimum operation yields the overlap region of the two sets and the maximum operation gives the highest membership in the overlap. The membership of the histogram in 'cold', indicated by the arrow in Fig. 9.8, is 0.73. By similar operations, the membership of the histogram in 'comfortable' and 'hot' are 0.40 and 0.00. It is interesting to note that there is no requirement that the sum of all memberships be 1.00.

9.5.3 Rule Application

The linguistic model of a process is commonly made of a series of ***if - then*** rules. These use the measured state of the process, the rule antecedents, to estimate the extent of control action, the rule consequents. Although each rule is simple, there must be a rule to cover every possible combination of fuzzy input values. Thus, the simplicity of the rules trades off against the number of rules. For complex systems the number of rules required may be very large.

The rules needed to describe a process are often obtained through consultation with workers who have expert knowledge of the process operation. These experts include the process designers, but more

importantly, the process operators. The rules can include both the normal operation of the process as well as the experience obtained through upsets and other abnormal conditions. Exception handling is a particular strength of fuzzy control systems.

For very complex systems, the experts may not be able to identify their thought processes in sufficient detail for rule creation. Rules may also be generated from operating data by searching for clusters in the input data space. A simple temperature control model can be constructed from the example of Fig. 9.7:

Rule 1 : IF (Temperature is Cold) THEN (Heater is On)

Rule 2 : IF (Temperature is Comfortable) THEN (Heater is Off)

Rule 3 : IF (Temperature is Hot) THEN (Heater is Off)

In Rule 1, (Temperature is Cold) is the membership value of the actual temperature in the 'cold' set. Rule 1 transfers the 0.66 membership in 'cold' to become 0.66 membership in the heater setting 'on'. Similar values from rules 2 and 3 are 0.33 and 0.00 in the 'off' setting for the heater. When several rules give membership values for the same output set, Mamdani used the maximum of the membership values. The result for the three rules is then 0.66 membership in 'on' and 0.33 membership in 'off'.

The rules presented in the above example are simple yet effective. To extend these to more complex control models, compound rules may be formulated. For example, if humidity was to be included in the room temperature control example, rules of the form: IF (Temperature is Cold) AND (Humidity is High) THEN (Heater is ON) might be used. Zadeh defined the logical operators as AND = Min (μ_A, μ_B) and OR = Max (μ_A, μ_B), where μ_A and μ_B are membership values in sets *A* and *B* respectively. In the above rule, the membership in 'on' will be the minimum of the two antecedent membership values. Zadeh also defined the NOT operator by assuming that complete membership in the set *A* is given by $\mu_A = 1$. The membership in NOT (*A*) is then given by μNOT (*A*) = $1 - \mu_A$. This gives the interesting result that *A* AND NOT (*A*) does not vanish, but gives a distribution corresponding to the overlap between *A* and its adjacent sets.

9.5.4 Defuzzification

The results of rule application are membership values in each of the consequent or output sets. These can be used directly where the membership values are viewed as the strength of the recommendations provided by the rules. It is possible that several outputs are recommended and some may be contradictory (e.g. heater on and heater off). In automatic control, one physical value of a controller output must be chosen from multiple recommendations. In decision support systems, there must be a consistent method to resolve conflict and define an appropriate compromise. Defuzzification is the process for converting fuzzy output values to a single value or final decision. Two methods are commonly used.

The first is the maximum membership method. All of the output membership functions are combined using the OR operator and the position of the highest membership value in the range of the output variable is used as the controller output. This method fails when there are two or more equal maximum membership values for different recommendations. Here the method becomes indecisive and does not produce a satisfactory result.

The second method uses the center of gravity of the combined output distribution to resolve this potential conflict and to consider all recommendations based on the strengths of their membership values. The center of gravity is given by $X_F = \dfrac{\int x\ (x)dx}{\int (x)dx}$ where x is a point in the output range and X_F is the final control value. These integrals are taken over the entire range of the output. By taking the center of gravity, conflicting rules essentially cancel and a fair weighting is obtained.

The output values used in the thermostat example are singletons. Singletons are fuzzy values with a membership of 1.00 at a single value rather than a membership function between 0 and 1 defined over an interval of values. In the example there were two, 'off' at 0% power and 'on' at 100% power. With singletons, the center of gravity equation integrals become a simple weighted average. Applying the rules gave $\mu_{ON} = 0.67$ and $\mu_{OFF} = 0.33$. Defuzzifying these gives a control output of 67% power. Although only two singleton output functions were used, with center of gravity defuzzification, the heater power decreases smoothly between fully on and fully off as the temperature increases between 10°C and 25°C.

In the histogram input case, applying the same rules gave $\mu_{ON} = 0.73$ and $\mu_{OFF} = 0.40$. Center of gravity defuzzification gave, in this case, a heater power of 65%. The sum of the membership functions was normalized by the denominator of the center of gravity calculation.

9.5.5 Conclusions

Linguistic descriptions in the form of membership functions and rules make up the model. The rules are generated *a priori* from expert knowledge or from data through system identification methods. Input membership functions are based on estimates of the vagueness of the descriptors used. Output membership functions can be initially set, but can be revised for controller tuning.

Once these are defined, the operating procedures for the calculations are well set out. Measurement data are converted to memberships through fuzzification procedures. The rules are applied using formalized operations to yield memberships in output sets. Finally, these are combined through defuzzification to give a final control output.

9.6 FUZZY LOGIC MODEL FOR GRADING OF APPLES

Agricultural produce is subject to quality inspection for optimum evaluation in the consumption cycle. Efforts to develop automated fruit classification systems have been increasing recently due to the drawbacks of manual grading such as subjectivity, tediousness, labor requirements, availability, cost and inconsistency.

However, applying automation in agriculture is not as simple as automating the industrial operations. There are two main differences. First, the agricultural environment is highly variable, in terms of weather, soil, etc. Second, biological materials, such as plants and commodities, display high variation due to their inherent morphological diversity. Techniques used in industrial applications, such as template matching and fixed object modeling are unlikely to produce satisfactory results in the classification or control of input from agricultural products. Therefore, self-learning techniques such as neural networks (*NN*) and fuzzy logic (*FL*) seem to represent a good approach.

Fuzzy logic can handle uncertainty, ambiguity and vagueness. It provides a means of translating qualitative and imprecise information into quantitative (linguistic) terms. Fuzzy logic is a non-parametric classification procedure, which can infer with nonlinear relations between input and output categories, maintaining flexibility in making decisions even on complex biological systems.

Fuzzy logic was successfully used to determine field trafficability, to decide the transfer of dairy cows between feeding groups, to predict the yield for precision farming, to control the start-up and shut-down of food extrusion processes, to steer a sprayer automatically, to predict corn breakage, to manage crop production, to reduce grain losses from a combine, to manage a food supply and to predict peanut maturity.

The main purpose of this study was to investigate the applicability of fuzzy logic to constructing and tuning fuzzy membership functions and to compare the accuracies of predictions of apple quality by a human expert and the proposed fuzzy logic model. Grading of apples was performed in terms of characteristics such as color, external defects, shape, weight and size. Readings of these properties were obtained from different measurement apparatuses, assuming that the same measurements can be done using a sensor fusion system in which measurements of features are collected and controlled automatically. The following objectives were included in this study:

1. To design a *FL* technique to classify apples according to their external features developing effective fuzzy membership functions and fuzzy rules for input and output variables based on quality standards and expert expectations.
2. To compare the classification results from the *FL* approach and from sensory evaluation by a human expert.
3. To establish a multi-sensor measuring system for quality features in the long term.

9.6.1 Apple Defects Used in the Study

No defect formation practices by applying forces on apples were performed. Only defects occurring naturally or forcedly on apple surfaces during the growing season and handling operations were accounted for in terms of number and size, ignoring their age. Scars, bitter pit, leaf roller, russeting, punctures and bruises were among the defects encountered on the surfaces of Golden Delicious apples. In addition to these defects, a size defect (lopsidedness) was also measured by taking the ratio of maximum height of the apple to the minimum height.

9.6.2 Materials and Methods

Five quality features, color, defect, shape, weight and size, were measured. Color was measured using a *CR*-200 Minolta colorimeter in the domain of L, a and b, where L is the lightness factor and a and b are the chromaticity coordinates. Sizes of surface defects (natural and bruises) on apples were determined using a special figure template, which consisted of a number of holes of different diameters. Size defects were determined measuring the maximum and minimum heights of apples using a Mitutoya electronic caliper. Maximum circumference measurement was performed using a Cranton circumference measuring device. Weight was measured using an electronic scale. Programming for fuzzy membership functions, fuzzification and defuzzification was done in Matlab.

The number of apples used was determined based on the availability of apples with quality features of the 3 quality groups (bad, medium and good). A total of 181 golden delicious apples were graded first by a human expert and then by the proposed fuzzy logic approach. The expert was trained on the external quality criteria for good, medium and bad apple groups defined by USDA standards (USDA, 1976). The USDA standards for apple quality explicitly define the quality criteria so that it is quite straightforward for an expert to follow up and apply them. Extremely large or small apples were already excluded by the handling personnel. Eighty of the apples were kept at room temperature for 4 days while another 80 were kept in a cooler (at about 3°C) for the same period to create color variation on the surfaces of apples. In addition, 21 of the apples were harvested before the others and kept for 15 days at room temperature for the same purpose of creating a variation in the appearance of the apples to be tested.

The Hue angle ($\tan^{-1}(b/a)$), which was used to represent the color of apples, was shown to be the best representation of human recognition of color. To simplify the problem, defects were collected under a single numerical value, "defect" after normalizing each defect component such as bruises, natural defects, russetting and size defects (lopsidedness).

$$\text{Defect} = 10 \times B + 5 \times ND + 3 \times R + 0.3 \times SD \quad ...(9.1)$$

where B is the amount of bruising, ND is the amount of natural defects, such as scars and leaf roller, as total area (normalized), R is the total area of russeting defect (normalized) and SD is the normalized size defect. Similarly, circumference, blush (reddish spots on the cheek of an apple) percentage and weight were combined under "Size" using the same procedure as with "Defect"

$$\text{Size} = 5 \times C + 3 \times W + 5 \times BL \quad ...(9.2)$$

where C is the circumference of the apple (normalized), W is weight (normalized) and BL is the normalized blush percentage. Coefficients used in the above equations were subjectively selected, based on the expert's expectations and USDA standards (USDA, 1976).

Although it was measured at the beginning, firmness was excluded from the evaluation, as it was difficult for the human expert to quantify it nondestructively. After the combinations of features given in the above equations, input variables were reduced to 3 defect, size and color. Along with the measurements of features, the apples were graded by the human expert into three quality groups, bad, medium and good, depending on the expert's experience, expectations and USDA standards (USDA, 1976). Fuzzy logic techniques were applied to classify apples after measuring the quality features. The grading performance of fuzzy logic proposed was determined by comparing the classification results from *FL* and the expert.

9.6.3 Application of Fuzzy Logic

Three main operations were applied in the fuzzy logic decision making process: selection of fuzzy inputs and outputs, formation of fuzzy rules, and fuzzy inference. A trial and error approach was used to develop membership functions. Although triangular and trapezoidal functions were used in establishing membership functions for defects and color (Fig. 9.9 and 9.10), an exponential function with the base of the irrational number e was used to simulate the inclination of the human expert in grading apples in terms of size (Fig. 9.11).

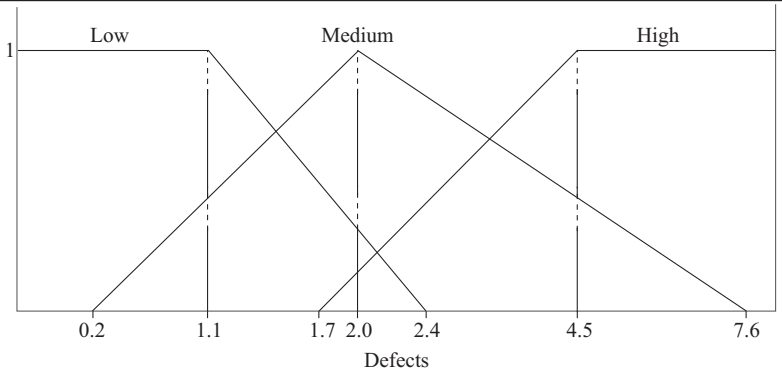

Fig. 9.9 Membership functions for the defect feature.

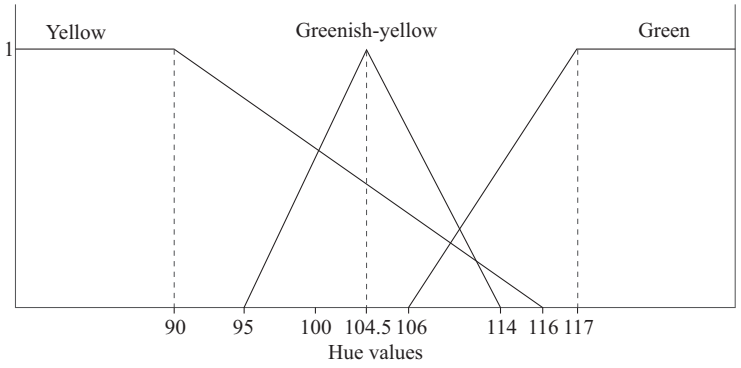

Fig. 9.10 Membership functions for the color feature.

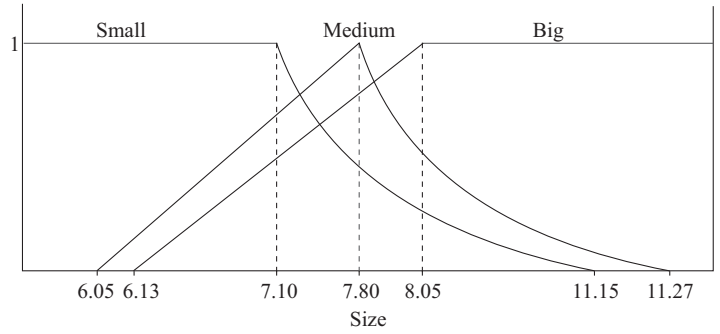

Fig. 9.11 Membership functions for the size feature.

$$\text{Size} = e^x \qquad \ldots(9.3)$$

where e is approximately 2.71828 and x is the value of size feature.

9.6.4 Fuzzy Rules

At this stage, human linguistic expressions were involved in fuzzy rules. The rules used in the evaluations of apple quality are given in Table 9.2. Two of the rules used to evaluate the quality of Golden Delicious apples are given below:

If the color is greenish, there is no defect, and it is a well formed large apple, then quality is very good (rule $Q_{1,1}$ in Table 9.2).

Table 9.2: Fuzzy rule tabulation

	$C_1 + S_1$	$C_1 + S_2$	$C_1 + S_3$	$C_2 + S_1$	$C_2 + S_2$	$C_2 + S_3$	$C_3 + S_1$	$C_2 + S_2$	$C_3 + S_3$
D_1	$Q_{1,1}$	$Q_{1,2}$	$Q_{2,3}$	$Q_{1,3}$	$Q_{2,5}$	$Q_{3,8}$	$Q_{2,6}$	$Q_{2,7}$	$Q_{3,15}$
D_2	$Q_{2,1}$	$Q_{2,2}$	$Q_{3,3}$	$Q_{2,4}$	$Q_{3,6}$	$Q_{3,9}$	$Q_{3,11}$	$Q_{3,13}$	$Q_{3,16}$
D_3	$Q_{3,1}$	$Q_{3,2}$	$Q_{3,4}$	$Q_{3,5}$	$Q_{3,7}$	$Q_{3,10}$	$Q_{3,12}$	$Q_{3,14}$	$Q_{3,17}$

Where, C_1 is the greenish color quality (desired), C_2 is greenish-yellow color quality medium), and C_3 is yellow color quality (bad); S_1, on the other hand, is well formed size (desired), S_2 is moderately formed size (medium), S_3 is badly formed size (bad). Finally, D_1 represents a low amount of defects (desired), while D_2 and D_3 represent moderate (medium) and high (bad) amounts of defects, respectively. For quality groups represented with "Q" in Table 1, the first subscript 1 stands for the best quality group, while 2 and 3 stand for the moderate and bad quality groups, respectively. The second subscript of Q shows the number of rules for the particular quality group, which ranges from 1 to 17 for the bad quality group.

If the color is pure yellow (overripe), there are a lot of defects, and it is a badly formed (small) apple, then quality is very bad (rule $Q_{3,17}$ in Table 9.2).

A fuzzy set is defined by the expression below:$\bar{}_C$

$$D = \{X.\ \mu_0(x))|\ x \in X\} \qquad ...(9.4)$$

$$\mu_0(x): \rightarrow [0, 1]$$

where X represents the universal set, D is a fuzzy subset in X and $\mu D(x)$ is the membership function of fuzzy set D. Degree of membership for any set ranges from 0 to1. A value of 1.0 represents a 100% membership while a value of 0 means 0% membership. If there are three subgroups of size, then three memberships are required to express the size values in a fuzzy rule.

Three primary set operations in fuzzy logic are AND, OR, and the Complement, which are given as follows

AND: $$\mu_C \wedge \mu_D = \min\ \{\mu_C, \mu_D\} \qquad ...(9.5)$$

OR: $$\mu_C \cup \mu_D = (\mu_C \vee \mu_D) = \max\ \{\mu_C, \mu_D\} \qquad ...(9.6)$$

$$\text{complement} = \quad = 1 - \mu_D \qquad ...(9.7)$$

The minimum method given by equation (9.5) was used to combine the membership degrees from each rule established. The minimum method chooses the most certain output among all the membership degrees. An example of the fuzzy AND (the minimum method) used in ***if-then*** rules to form the Q_{11} quality group in Table 9.2 is given as follows;

$$Q_{11} = (C_1 \wedge S_1 \wedge D_1) = \min \{C_1, S_1, D_1\} \quad ...(9.8)$$

On the other hand, the fuzzy OR (the maximum method) rule was used in evaluating the results of the fuzzy rules given in Table 9.2; determination of the quality group that an apple would belong to, for instance, was done by calculating the most likely membership degree using equations 9.9 through 9.13. If,

$$k_1 = (Q_{1,1}, Q_{1,2}, Q_{1,3}) \quad ...(9.9)$$

$$k_2 = (Q_{2,1}, Q_{2,2}, Q_{2,3}\ Q_{2,4}, Q_{2,5}, Q_{2,6}) \quad ...(9.10)$$

$$k_3 = (Q_{3,1}, Q_{3,2}, Q_{3,3}\ Q_{3,4}, Q_{3,5},$$
$$Q_{3,6}, Q_{3,7}, Q_{3,8}, Q_{3,9}, Q_{3,10}, Q_{3,11}, Q_{3,12}, Q_{3,13}, Q_{3,14}, Q_{3,15}, Q_{3,16}, Q_{3,17}) \quad ...(9.11)$$

where k is the quality output group that contains different class membership degrees and the output vector y given in equation 10 below determines the probabilities of belonging to a quality group for an input sample before defuzzification:

$$y = [\max(k_1) \max(k_2) \max(k_3)] \quad ...(9.12)$$

where, for example,

$$\max(k_1) = (Q_{1,1} \vee Q_{1,2} \vee Q_{1,3}) = \max \{Q_{1,1,}\ Q_{1,2}, Q_{1,3}\} \quad ...(9.13)$$

then, equation 11 produces the membership degree for the best class (Lee, 1990).

9.6.5 Determination of Membership Functions

Membership functions are in general developed by using intuition and qualitative assessment of the relations between the input variable(s) and output classes. In the existence of more than one membership function that is actually in the nature of the fuzzy logic approach, the challenge is to assign input data into one or more of the overlapping membership functions. These functions can be defined either by linguistic terms or numerical ranges, or both. The membership function used in this study for defect quality in general is given in equation 9.4. The membership function for high amounts of defects, for instance, was formed as given below:

If the input vector x is given as x = [defects, size, color], then the membership function for the class of a high amount of defects (D_3) is

$$\mu(D_3) = 0, \text{ when } x(1) < 1.75$$

$$\mu(D_3) = \frac{(x(1) - 1.75)}{2.77}, \text{ when } 1.75 \le x(1) \le 4.52 \text{ or} \quad ...(9.14)$$

$$\mu(D_3) = 1, \text{ when } x(1) \ge 4.52$$

For a medium amount of defects (D_2), the membership function is

$$\mu(D_2) = 0, \text{ when defect innput } x(1) < 0.24 \text{ or } x(1) > 7.6$$

$$\mu(D_2) = \qquad\qquad , \text{ when } 0.24 \le x(1) \le 2 \quad ...(9.15)$$

$$\mu(D_2) = \frac{(7.6 - x(1))}{5.6}, \text{ when } 2 \le x(1) \le 7.6$$

For a low amount of defects (D_1), the membership function is

$$\mu(D_1) = 0, \text{ when defect input } x(1) > 0.24$$

$$\mu(D_1) = \frac{(0.24 - x(1))}{1.3}, \text{ when } 1.1 \le x(1) \le 2.4 \qquad ...(9.16)$$

$$\mu(D_1) = 1, \text{ when } x(1) \le 1.1.$$

Calculations for the quality groups of color and size were performed using the same approach as defect. Three membership functions for the quality classes of defect, color and size are schematically shown in Fig. 9.9, 9.10 and 9.11, respectively.

9.6.6 Defuzzification

The centroid method, which is also known as the center of mass, was used for defuzzification. After execution of the rules established and shown in the previous section, the output grades described below are obtained.

For low quality, degree of membership is calculated using equation 9.17.

$$SC = \frac{[(y(3) \times (c + 1.5))]}{2} \qquad ...(9.17)$$

where,

$$c = c_2 - c_1$$

$$c_1 = [0.5 \times y(3) + 0.75]$$

$$c_2 = [2.25 - (0.5 \times y(3))]$$

y (3) is the low quality output from the output vector y (equation 9.12), c, c_1, and c_2 are shown in Fig. 9.12 and sc is the area of the trapezoid formed. Membership degrees of sb and sa are calculated using the same approach in equation 15 for the medium and high quality classes.

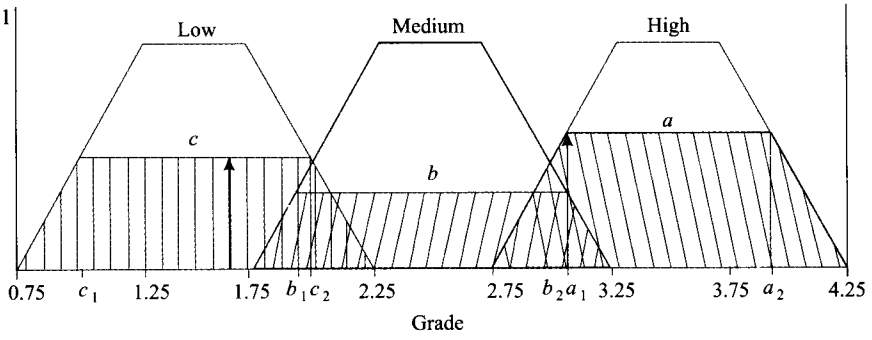

Fig. 9.12 Membership functions for output quality groups and determination of a grade for an apple.

In the defuzzification stage, the overall grade for a particular apple was found by taking the average of the weighted possible outputs using the weighted average method. Equation 9.18 was used for this purpose

$$wa = \frac{sa \times 3.5 + sb \times 2.5 + sc \times 1.5}{sa + sb + sc} \quad ...(9.18)$$

where wa is the weighted average for the grade of a particular apple. From each of the 3 output categories, trapezoidal areas (sa, sb, and sc) were calculated for the apple being graded. Then, the weighted average of the 3 trapezoidal areas shown in Fig. 9.12 was calculated to find the final grade for the particular apple.

9.6.7 Results and Discussion

In the results of the defuzzification process, grades for all the apples were calculated between 0 and 3.99. Grade (g) ranges for the output quality classes were chosen as follows: $2.3 \leq g \leq 4$ for the best class, $1.4 \leq g < 2.3$ for the moderate class and $0 \leq g < 1.4$ for the bad class. The resulting classification accuracies obtained from fuzzy logic are given in Table 9.3 in comparison with the classification results from the expert. Fuzzy logic predicted around 89% of apples correctly (Table 9.3). Misclassification errors observed were among adjacent groups in general. This kind of error is usually acceptable.

Table 9.3: Comparison if fuzzy logic and human expert in classification of apples.

		Fuzzy logic prediction				
	Class	1	2	3	Total predicted	%
Human Expert	1	121	9	0	130	93.1
	2	6	33	2	41	80.5
	3	1	2	7	10	70.0
Total Observed		128	44	9	161*/181	
%		94.5	75.0	77.8		89.0

*Number of apples correctly classified by fuzzy logic.

Determination of membership functions in terms of shape and boundary has a clear effect on the result of classification performed by fuzzy logic. This situation greatly depends on experience and knowledge. Finding the right shape and the boundaries for the membership function will increase the accuracy of the fuzzy logic application. Statistics of the class populations, such as average, standard deviation and minimum-maximum values, could help the determination of membership functions. Therefore, parameters of fuzzy logic, such as function shape, threshold, which is to determine the overlapping amount and condition among the membership functions, input and output levels, and function rules, must be tested to find the optimum classification result. These application criterions of fuzzy logic that must be investigated are both disadvantageous as it takes time to apply all the alternatives, and powerful as it provides an opportunity to build a system compatible with the standards and expectations.

9.6.8 Conclusion

Fuzzy logic was successfully applied to serve as a decision support technique in grading apples. Grading results obtained from fuzzy logic showed a good general agreement with the results from the human expert, providing good flexibility in reflecting the expert's expectations and grading standards into the results. It was also seen that color, defects and size are three important criteria in apple classification. However, variables such as firmness, internal defects and some other sensory evaluations, in addition to the features mentioned earlier, could increase the efficiency of decisions made regarding apple quality.

9.7 AN INTRODUCTORY EXAMPLE: FUZZY V/S NON-FUZZY

To illustrate the value of fuzzy logic, fuzzy and non-fuzzy approaches are applied to the same problem. First the problem is solved using the conventional (non-fuzzy) method, writing MATLAB commands that spell out linear and piecewise-linear relations. Then, the same system is solved using fuzzy logic.

Consider the tipping problem: what is the "right" amount to tip your waitperson? Given a number between 0 and 10 that represents the quality of service at a restaurant (where 10 is excellent), what should the tip be?

This problem is based on tipping as it is typically practiced in the United States. An average tip for a meal in the U.S. is 15%, though the actual amount may vary depending on the quality of the service provided.

9.7.1 The Non-Fuzzy Approach

Let's start with the simplest possible relationship (Fig. 9.13). Suppose that the tip always equals 15% of the total bill.

tip = 0.15

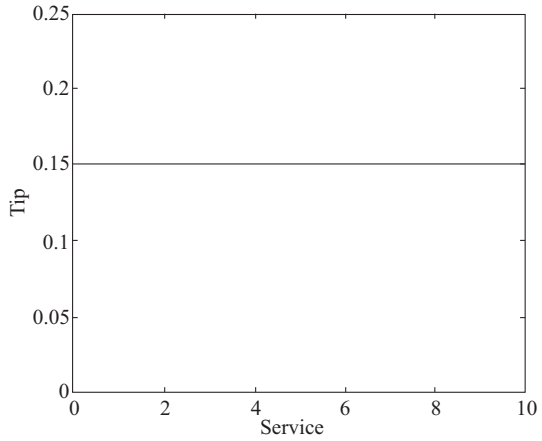

Fig. 9.13 Constant tipping.

This does not really take into account the quality of the service, so we need to add a new term to the equation. Since service is rated on a scale of 0 to 10, we might have the tip go linearly from 5% if the service is bad to 25% if the service is excellent (Fig. 9.14). Now our relation looks like this:

tip = 0.20/10 * service + 0.05

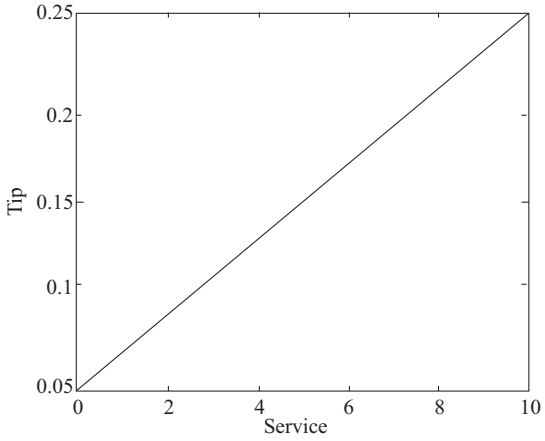

Fig. 9. 14 Linear tipping.

The formula does what we want it to do, and it is pretty straight forward. However, we may want the tip to reflect the quality of the food as well. This extension of the problem is defined as follows:

Given two sets of numbers between 0 and 10 (where 10 is excellent) that respectively represent the quality of the service and the quality of the food at a restaurant, what should the tip be? Let's see how the formula will be affected now that we've added another variable (Fig. 9.15). Suppose we try:

tip = 0.20/20 × (service + food) + 0.05

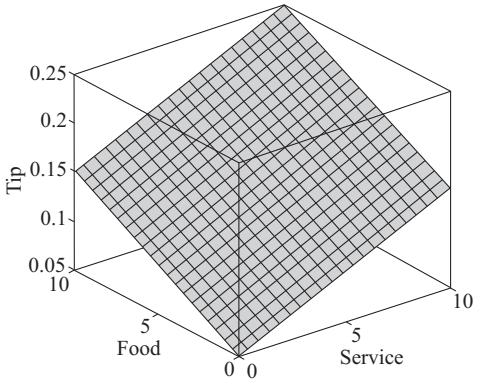

Fig. 9.15 Tipping depend on service and quality of food.

In this case, the results look pretty, but when you look at them closely, they do not seem quite right. Suppose you want the service to be a more important factor than the food quality. Let's say that the service will account for 80% of the overall tipping "grade" and the food will make up the other 20%. Try:

servRatio = 0.8;

tip= servRatio × (0.20/10 × service + 0.05) + (1– servRatio) × (0.20/10 × food + 0.05);

The response is still somehow too uniformly linear. Suppose you want more of a flat response in the middle, *i.e.,* you want to give a 15% tip in general, and will depart from this plateau only if the service is exceptionally good or bad (Fig. 9.16).

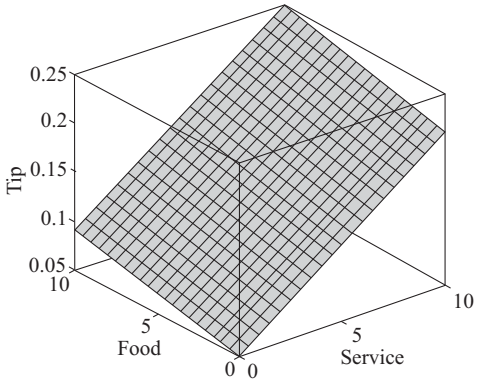

Fig. 9.16 Tipping based on the service to be a more important factor than the food quality.

This, in turn, means that those nice linear mappings no longer apply. We can still salvage things by using a piecewise linear construction (Fig. 9.17). Let's return to the one-dimensional problem of just considering the service. You can string together a simple conditional statement using breakpoints like this:

if service < 3,

tip = (0.10/3) × service + 0.05;

else if service < 7 ,

tip = 0.15;

else if service < =10,

tip = (0.10/3) × (service –7) + 0.15;

end

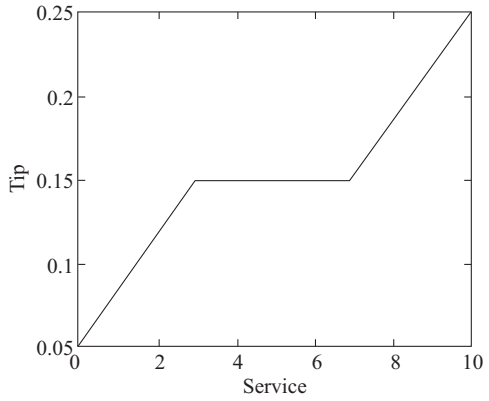

Fig. 9. 17 Tipping using a piecewise linear construction.

If we extend this to two dimensions (Fig. 9.18), where we take food into account again, something like this result:

servRatio = 0.8;

if service < 3,

tip = ((0.10/3) × service + 0.05) × servRatio + (1 – servRatio) × (0.20/10 × food + 0.05);

else if service < 7,

tip = (0.15) × servRatio + (1 – servRatio) × (0.20/10 × food + 0.05);

else,

tip = ((0.10/3) × (service – 7) + 0.15) × servRatio + (1 – servRatio) × (0.20/10 × food + 0.05);

end

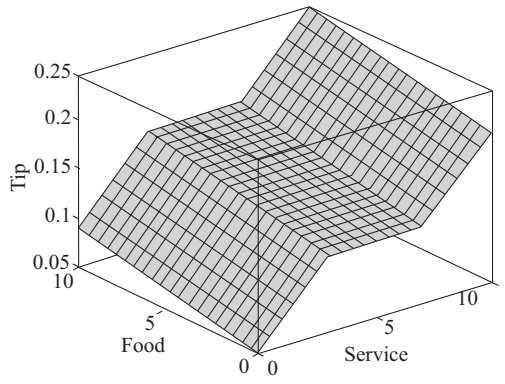

Fig. 9.18 Tipping with two-dimensional variation.

The plot looks good, but the function is surprisingly complicated. It was a little tricky to code this correctly, and it is definitely not easy to modify this code in the future. Moreover, it is even less apparent how the algorithm works to someone who did not witness the original design process.

9.7.2 The Fuzzy Approach

It would be nice if we could just capture the essentials of this problem, leaving aside all the factors that could be arbitrary. If we make a list of what really matters in this problem, we might end up with the following rule descriptions:

1. *If service is poor, then tip is cheap*
2. *If service is good, then tip is average*
3. *If service is excellent, then tip is generous*

The order in which the rules are presented here is arbitrary. It does not matter which rules come first. If we wanted to include the food's effect on the tip, we might add the following two rules:

4. *If food is rancid, then tip is cheap*
5. *If food is delicious, then tip is generous*

In fact, we can combine the two different lists of rules into one tight list of three rules like so:

1. *If service is poor or the food is rancid, then tip is cheap*
2. *If service is good, then tip is average*
3. *If service is excellent or food is delicious, then tip is generous*

These three rules are the core of our solution. And coincidentally, we have just defined the rules for a fuzzy logic system. Now if we give mathematical meaning to the linguistic variables (what is an "average" tip, for example?) we would have a complete fuzzy inference system. Of course, there's a lot left to the methodology of fuzzy logic that we're not mentioning right now, things like:

- How are the rules all combined?
- How do I define mathematically what an "average" tip is?

The details of the method do not really change much from problem to problem - the mechanics of fuzzy logic are not terribly complex. What matters is what we have shown in this preliminary exposition: fuzzy is adaptable, simple, and easily applied.

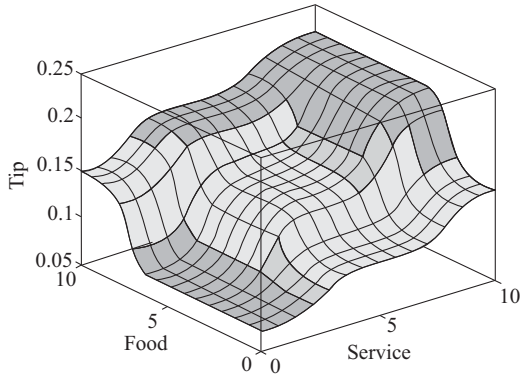

Fig. 9.19 Tipping using fuzzy logic.

Here is the picture associated with the fuzzy system that solves this problem (Fig. 9.19). The picture above was generated by the three rules above.

9.7.3 Some Observations

Here are some observations about the example so far. We found a piecewise linear relation that solved the problem. It worked, but it was something of a nuisance to derive, and once we wrote it down as code, it was not very easy to interpret. On the other hand, the fuzzy system is based on some "common sense" statements. Also, we were able to add two more rules to the bottom of the list that influenced the shape of the overall output without needing to undo what had already been done. In other words, the subsequent modification was pretty easy.

Moreover, by using fuzzy logic rules, the maintenance of the structure of the algorithm decouples along fairly clean lines. The notion of an average tip might change from day to day, city to city, country to country, but the underlying logic the same: if the service is good, the tip should be average. You can recalibrate the method quickly by simply shifting the fuzzy set that defines average without rewriting the fuzzy rules.

You can do this sort of thing with lists of piecewise linear functions, but there is a greater likelihood that recalibration will not be so quick and simple. For example, here is the piecewise linear tipping problem slightly rewritten to make it more generic. It performs the same function as before, only now the constants can be easily changed.

```
% Establish constants
lowTip=0.05; averTip=0.15; highTip=0.25;
tipRange=highTip–lowTip;
badService=0; okayService=3;
goodService=7; greatService=10;
serviceRange=greatService–badService;
badFood=0; greatFood=10;
foodRange=greatFood–badFood;
% If service is poor or food is rancid, tip is cheap
if service<okayService,
tip=(((averTip–lowTip)/(okayService–badService)) ...
*service+lowTip)*servRatio + ...
(1–servRatio)*(tipRange/foodRange*food+lowTip);
% If service is good, tip is average
elseif service<goodService,
tip=averTip*servRatio + (1–servRatio)* ...
(tipRange/foodRange*food+lowTip);
```

```
% If service is excellent or food is delicious, tip is generous
else,
tip=(((highTip–averTip)/ ...
(greatService–goodService))* ...
(service–goodService)+averTip)*servRatio + ...
(1–servRatio)*(tipRange/foodRange*food+lowTip);
end
```

Notice the tendency here, as with all code, for creeping generality to render the algorithm more and more opaque, threatening eventually to obscure it completely. What we are doing here is not that complicated. True, we can fight this tendency to be obscure by adding still more comments, or perhaps by trying to rewrite it in slightly more self-evident ways, but the medium is not on our side.

The truly fascinating thing to notice is that if we remove everything except for three comments, what remain are exactly the fuzzy rules we wrote down before:

```
% If service is poor or food is rancid, tip is cheap
% If service is good, tip is average
% If service is excellent or food is delicious, tip is generous
```

If, as with a fuzzy system, the comment is identical with the code, think how much more likely your code is to have comments! Fuzzy logic lets the language that's clearest to you, high level comments, also have meaning to the machine, which is why it is a very successful technique for bridging the gap between people and machines.

QUESTION BANK

1. Why use fuzzy logic?
2. What are the applications of fuzzy logic?
3. When not use fuzzy logic?
4. Compare non-fuzzy logic and fuzzy logic approaches.

REFERENCES

1. L.A. Zadeh, Fuzzy sets, *Information and Control*, Vol. 8, pp. 338-353, 1965.
2. USDA Agricultural Marketing Service, United States Standards for Grades of Apples, Washington, D.C., 1976.
3. W.J.M. Kickert and H.R. Van Nauta Lemke, Application of a fuzzy controller in a warm water plat, *Automatica*, Vol. 12, No. 4, pp. 301-308, 1976.
4. C.P. Pappis and E.H. Mamdani, A fuzzy logic controller for a traffic junction, *IEEE Transactions on Systems*, *Man and Cybernetics*, Vol. 7, No. 10, pp. 707-717, 1977.

5. M. Sugeno and M. Nishida, Fuzzy control of model car, *Fuzzy Sets and Systems*, Vol. 16, No. 2, pp. 103-113, 1985.
6. B.P. Graham and R.B. Newell, Fuzzy identification and control of a liquid level rig, *Fuzzy Sets and Systems*, Vol. 26, No. 3, pp. 255-273, 1988.
7. E. Czogala and T. Rawlik, Modeling of a fuzzy controller with application to the control of biological processes, *Fuzzy Sets and Systems*, Vol. 31, No. 1, pp. 13-22, 1989.
8. C.C. Lee, Fuzzy logic in control systems: Fuzzy logic controller- Part I and Part II, *IEEE Transactions on Systems*, *Man and Cybernetics*, 20: 404-435, 1990.
9. S. Thangavadivelu and T.S. Colvin, Trafficability determination using fuzzy set theory, *Transactions of the ASAE*, Vol. 34, No. 5, pp. 2272- 2278, 1991.
10. T. Tobi and T. Hanafusa, A practical application of fuzzy control for an air-conditioning system, *International Journal of Approximate Reasoning*, Vol. 5, No. 3, pp. 331-348, 1991.
11. U. Ben-Hannan, K. Peleg and P.O. Gutman, Classification of fruits by a Boltzman perceptron neural network, *Automatica*, Vol. 28, pp. 961-968, 1992.
12. R. Palm, Control of a redundant manipulator using fuzzy rules, *Fuzzy Sets and Systems*, Vol. 45, No. 3, pp. 279-298, 1992.
13. Q. Yang, Classification of apple surface features using machine vision and neural networks, Computer, Electron. *Agriculture*, Vol. 9, pp. 1-12, 1993.
14. J.J. Song and S. Park, A *Fuzzy Dynamic Learning Controller for Chemical Process Control*, Vol. 54, No. 2, pp. 121-133, 1993.
15. S. Kikuchi, V. Perincherry, P. Chakroborty and H. Takahasgi, Modeling of driver anxiety during signal change intervals, Transportation research record, No. 1339, pp. 27-35, 1993.
16. N. Kiupel and P.M. Frank, Fuzzy control of steam turbines, *International Journal of Systems Science*, Vol. 24, No. 10, pp. 1905-1914, 1993.
17. T.S. Liu and J.C. Wu, A model for rider-motorcycle system using fuzzy control, *IEEE Transactions on Systems*, *Man and Cybernetics*, Vol. 23, No. 1, pp. 267-276, 1993.
18. J.R. Ambuel, T.S. Colvin and D.L. Karlen, A fuzzy logic yield simulator for prescription farming, *Transactions of the ASAE*, Vol. 37, No. 6, pp. 1999-2009, 1994.
19. A. Hofaifar, B. Sayyarodsari and J.E. Hogans, Fuzzy controller robot arm trajectory, *Information Sciences*: *Applications*, Vol. 2, No. 2, pp. 69-83, 1994.
20. S. Chen and E.G. Roger, Evaluation of cabbage seedling quality by fuzzy logic, ASAE Paper No. 943028, St. Joseph, MI, 1994.
21. P. Grinspan, Y. Edan, E.H. Kahn and E. Maltz, A fuzzy logic expert system for dairy cow transfer between feeding groups, *Transactions of the ASAE*, Vol. 37, and No. 5, pp. 1647-1654, 1994.
22. P.L. Chang and Y.C. Chen, A fuzzy multi-criteria decision making method for technology transfer strategy selection in biotechnology, *Fuzzy Sets and Systems*, Vol. 63, No. 2, pp. 131-139, 1994.
23. A. Marell and K. Westin, Intelligent Transportation System and Traffic Safety Drivers Perception and Acceptance of Electronic Speed Checkers, *Transportation Research Part C*, Vol. 7 pp. 131-147, USA, 1999.

24. D. Teodorovic, Fuzzy Logic Systems for Transportation Engineering: The State Of The Art, *Transportation Research* Part A, Vol. 33, pp. 337-364, USA, 1999.
25. R. Elvik, How much do road accidents cost the national economy, *Accident Analysis and Prevention*, Volume: 32, pp: 849-851, 2000.
26. M.A. Shahin, B.P. Verma, and E.W. Tollner, Fuzzy logic model for predicting peanut maturity, *Transactions of the ASAE*, Vol. 43, No. 2, pp. 483-490, 2000.

CHAPTER 10

Neural Networks Fundamentals

10.1 INTRODUCTION

The artificial neural networks, which we describe in this course, are all variations on the parallel distributed processing (PDP) idea. The architecture of each network is based on very similar building blocks, which perform the processing. In this chapter we first discuss these processing units and discuss different network topologies. Learning strategies as a basis for an adaptive system will be presented in the last section.

10.2 BIOLOGICAL NEURAL NETWORK

The term 'neural network' comes from the intended analogy with the functioning of the human brain adopting simplified models of 'biological neural network'. The human brain consists of nearly 1011 neurons (nerve cells) of different types. In a typical neuron, one can find nucleus with which the connections with other neurons are made through a network of fibres called dendrites. Extending out from the nucleus is the axon, which transmits, by means of complex chemical process, electric potentials to the neurons, with which the axon is connected to (Fig. 10.1). When signals, received by neuron, become equal or surpass their threshold values, it 'triggers' sending an electric signal of constant level and duration through axon. In this way, the message is transferred from one neuron to the other.

In the neural network, the neurons or the processing units may have several input paths corresponding to the dendrites. The units combine usually by a simple summation, that is, the weighted values of these paths (Fig. 10.2). The weighted value is passed to the neuron, where it is modified by threshold function such as sigmoid function. The modified value is directly presented to the next neuron.

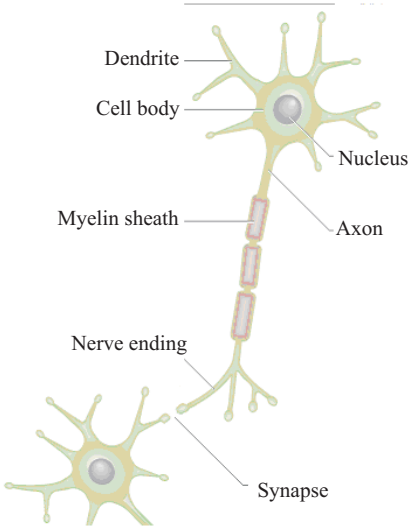

Fig. 10.1 Schematic representation of biological neuron network.

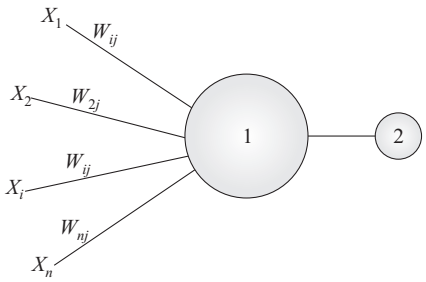

Fig. 10.2 Schematic representation of mathematical neuron network.

10.3 A FRAMEWORK FOR DISTRIBUTED REPRESENTATION

An artificial network consists of a pool of simple processing units, which communicate by sending signals to each other over a large number of weighted connections. A set of major aspects of a parallel distributed model can be distinguished as:

- a set of processing units (‘neurons’, ‘cells’);
- a state of activation y_k for every unit, which is equivalent to the output of the unit;
- connections between the units. Generally each connection is defined by a weight *wjk* which determines the effect which the signal of unit *j* has on unit *k*;
- a propagation rule, which determines the effective input s_k of a unit from its external inputs;
- an activation function F_k, which determines the new level of activation based on the effective input *sk*(*t*) and the current activation $y_k(t)$ (i.e., the update);’

- an external input (aka bias, offset) θ_k for each unit;
- a method for information gathering (the learning rule);
- an environment within which the system must operate, providing input signals and if necessary error signals.

Figure 10.3 illustrates these basics, some of which will be discussed in the next sections.

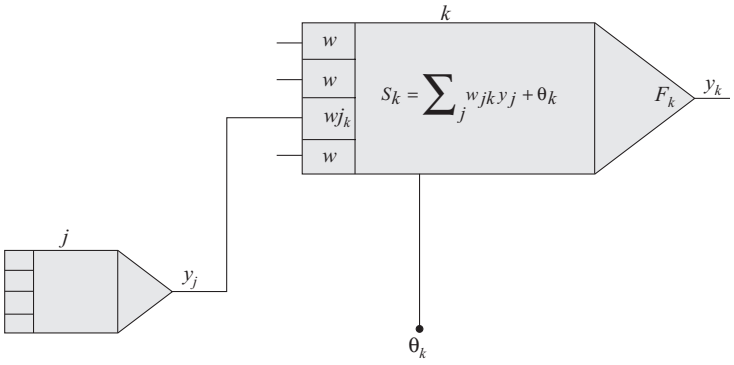

Fig. 10.3 The basic components of an artificial neural network. The propagation rule used here is the 'standard' weighted summation.

10.3.1 Processing Units

Each unit performs a relatively simple job: receive input from neighbors or external sources and use this to compute an output signal, which is propagated to other units. Apart from this processing, a second task is the adjustment of the weights. The system is inherently parallel in the sense that many units can carry out their computations at the same time.

Within neural systems it is useful to distinguish three types of units: input units (indicated by an index *i*) which receive data from outside the neural network, output units (indicated by an index *o*) which send data out of the neural network, and hidden units (indicated by an index *h*) whose input and output signals remain within the neural network.

During operation, units can be updated either synchronously or asynchronously. With synchronous updating, all units update their activation simultaneously; with asynchronous updating, each unit has a (usually fixed) probability of updating its activation at a time *t*, and usually only one unit will be able to do this at a time. In some cases the latter model has some advantages.

10.3.2 Connections between Units

In most cases we assume that each unit provides an additive contribution to the input of the unit with which it is connected. The total input to unit *k* is simply the weighted sum of the separate outputs from each of the connected units plus a bias or offset term θ_k:

$$s_k(t) = \sum_j w_{jk}(t)\, y_j(t) + \theta_k(t) \qquad ...(10.1)$$

The contribution for positive w_{jk} is considered as an excitation and for negative w_{jk} as inhibition. In some cases more complex rules for combining inputs are used, in which a distinction is made between excitatory and inhibitory inputs. We call units with a propagation rule (10.1) sigma units.

A different propagation rule, introduced by Feldman and Ballard, is known as the propagation rule for the sigma-pi unit is given by

$$s_k(t) = \sum_j w_{jk}(t) \prod_m y_{jm}(t) + \theta_k(t) \qquad ...(10.2)$$

Often, the y_{jm} are weighted before multiplication. Although these units are not frequently used, they have their value for gating of input, as well as implementation of lookup tables.

10.3.3 Activation and Output Rules

We also need a rule, which gives the effect of the total input on the activation of the unit. We need a function F_k which takes the total input $s_k(t)$ and the current activation $y_k(t)$ and produces a new value of the activation of the unit k:

$$y_k(t+1) = F_k(y_k(t), s_k(t)). \qquad ...(10.3)$$

Often, the activation function is a non-decreasing function of the total input of the unit:

$$y_k(t+1) = F_k(s_k(t)) = F_k\left(\sum_j w_{jk}(t)\, y_j(t) + \theta_k(t)\right) \qquad ...(10.4)$$

although activation functions are not restricted to non-decreasing functions. Generally, some sort of threshold function is used: a hard limiting threshold function (a sgn function), or a linear or semi-linear function, or a smoothly limiting threshold (see Fig. 10.4). For this smoothly limiting function often a sigmoid (S-shaped) function like

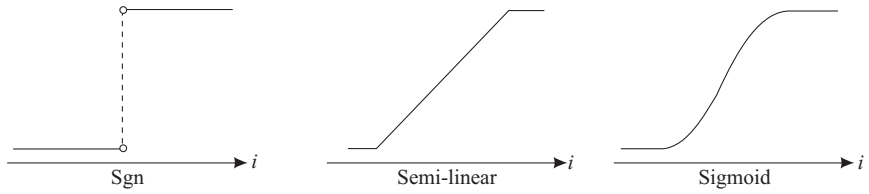

Fig. 10.4 Various activation functions for a unit.

$$y_k = F(s_k) = \frac{1}{1 + e^{-s_k}} \qquad ...(10.5)$$

is used. In some applications a hyperbolic tangent is used, yielding output values in the range [–1; +1].

In some cases, the output of a unit can be a stochastic function of the total input of the unit. In that case the activation is not deterministically determined by the neuron input, but the neuron input determines the probability p that a neuron get a high activation value:

$$p(y_k \leftarrow 1) = \frac{1}{1 + e^{-s_k/T}} \qquad ...(10.6)$$

in which T (temperature) is a parameter which determines the slope of the probability function.

In all networks we consider that the output of a neuron is to be identical to its activation level.

10.4 NETWORK TOPOLOGIES

In the previous section we discussed the properties of the basic processing unit in an artificial neural network. This section focuses on the pattern of connections between the units and the propagation of data.

As for this pattern of connections, the main distinction we can make is between:

- Feed-forward networks, where the data flow from input to output units is strictly feed-forward. The data processing can extend over multiple (layers of) units, but no feedback connections are present, that is, connections extending from outputs of units to inputs of units in the same layer or previous layers.
- Recurrent networks that do contain feedback connections. Contrary to feed-forward networks, the dynamical properties of the network are important. In some cases, the activation values of the units undergo a relaxation process such that the network will evolve to a stable state in which these activations do not change anymore. In other applications, the change of the activation values of the output neurons are significant, such that the dynamical behavior constitutes the output of the network.

Classical examples of feed-forward networks are the Perceptron and Adaline, which will be discussed in the next chapter. Examples of recurrent networks have been presented by Anderson, Kohonen, and Hopfield and will be discussed in subsequent chapters.

10.5 TRAINING OF ARTIFICIAL NEURAL NETWORKS

A neural network has to be configured such that the application of a set of inputs produces (either 'direct' or via a relaxation process) the desired set of outputs. Various methods to set the strengths of the connections exist. One way is to set the weights explicitly, using a priori knowledge. Another way is to 'train' the neural network by feeding it teaching patterns and letting it change its weights according to some learning rule.

10.5.1 Paradigms of Learning

We can categorize the learning situations in two distinct sorts. These are:

- Supervised learning or Associative learning in which the network is trained by providing it with input and matching output patterns. These input-output pairs can be provided by an external teacher, or by the system, which contains the network (self-supervised).
- Unsupervised learning or Self-organization in which an (output) unit is trained to respond to clusters of pattern within the input. In this paradigm the system is supposed to discover

statistically salient features of the input population. Unlike the supervised learning paradigm, there is no a priori set of categories into which the patterns are to be classified rather the system must develop its own representation of the input stimuli.

10.5.2 Modifying Patterns of Connectivity

Both learning paradigms discussed above result in an adjustment of the weights of the connections between units, according to some modification rule. Virtually all learning rules for models of this type can be considered as a variant of the Hebbian learning rule. The basic idea is that if two units j and k are active simultaneously, their interconnection must be strengthened. If j receives input from k, the simplest version of Hebbian learning prescribes to modify the weight w_{jk} with

$$\Delta w_{jk} = \gamma y_j y_k \qquad ...(10.7)$$

where γ is a positive constant of proportionality representing the learning rate. Another common rule uses not the actual activation of unit k but the difference between the actual and desired activation for adjusting the weights:

$$\Delta w_{jk} = \gamma y_j (d_k - y_k) \qquad ...(10.8)$$

in which d_k is the desired activation provided by a teacher. This is often called the Widrow-Hoff rule or the delta rule, and will be discussed in the next chapter.

Many variants (often very exotic ones) have been published the last few years. In the next chapters some of these update rules will be discussed.

10.6 NOTATION AND TERMINOLOGY

10.6.1 Notation

We use the following notation in our formulae. Note that not all symbols are meaningful for all networks, and that in some cases subscripts or superscripts may be left out (e.g., p is often not necessary) or added (e.g., vectors can, contrariwise to the notation below, have indices) where necessary. Vectors are indicated with a bold non-slanted font:

j, k,…. the unit j, k, …;

i an input unit;

h a hidden unit;

o an output unit;

x^p the pth input pattern vector;

x_j^p the jth element of the pth input pattern vector;

s^p the input to a set of neurons when input pattern vector p is clamped (i.e., presented to the network); often: the input of the network by clamping input pattern vector p;

d^p the desired output of the network when input pattern vector p was input to the network;

d_j^p the jth element of the desired output of the network when input pattern vector p was input to the network;

y^p the activation values of the network when input pattern vector p was input to the network;
y_j^p the activation values of element j of the network when input pattern vector p was input to the network;
W the matrix of connection weights;
w_j the weights of the connections which feed into unit j;
w_{jk} the weight of the connection from unit j to unit k;
F_j the activation function associated with unit j;
γ_{jk} the learning rate associated with weight w_{jk};
θ the biases to the units;
θ_j the bias input to unit j;
U_j the threshold of unit j in F_j;
E_p the error in the output of the network when input pattern vector p is input;
ε the energy of the network.

10.6.2 Terminology

Output vs. activation of a unit. Since there is no need to do otherwise, we consider the output and the activation value of a unit to be one and the same thing. That is, the output of each neuron equals its activation value.

Bias, offset, threshold: These terms all refer to a constant (i.e., independent of the network input but adapted by the learning rule) term which is input to a unit. They may be used interchangeably, although the latter two terms are often envisaged as a property of the activation function. Furthermore, this external input is usually implemented (and can be written) as a weight from a unit with activation value 1.

Number of layers: In a feed-forward network, the inputs perform no computation and their layer is therefore not counted. Thus a network with one input layer, one hidden layer, and one output layer is referred to as a network with two layers. This convention is widely though not yet universally used.

Representation vs. learning: When using a neural network one has to distinguish two issues which influence the performance of the system. The first one is the representational power of the network, the second one is the learning algorithm.

The representational power of a neural network refers to the ability of a neural network to represent a desired function. Because a neural network is built from a set of standard functions, in most cases the network will only approximate the desired function, and even for an optimal set of weights the approximation error is not zero.

The second issue is the learning algorithm. Given that there exist a set of optimal weights in the network, is there a procedure to (iteratively) find this set of weights?

QUESTION BANK

1. What are the major aspects of parallel distributed model?
2. Explain the biological neural network.
3. What are the basic components of artificial neural network?
4. What are the network topologies?
5. What are the various activation function? Explain them schematically.
6. What are the paradigms of neural network learning?

REFERENCES

1. D.O. Hebb, *The Organization of Behaviour*, New York: Wiley, 1949.
2. B. Widrow, *Generalization and Information Storage in Networks of Adaline Neurons*, *in Self Organizing Systems* 1962, ed. M.C. Jovitz, G.T. Jacobi, G. Goldstein, Washington, D.C: Spartan Books, PP. 435-461, 1962.
3. J.A. Anderson, Neural models with cognitive implications. In D. LaBerge and S.J. Samuels (Eds.), *Basic Processes in Reading Perception and Comprehension Models* (pp. 27-90). Hillsdale, NJ: Erlbaum, 1977.
4. T. Kohonen, Associative Memory: *A System-Theoretical Approach*, Springer-Verlag, 1977.
5. J.A. Feldman, and D.H. Ballard, Connectionist models and their properties, Cognitive Science, Vol. 6, pp. 205-254, 1982.
6. J.J. Hopfield,. Neural networks and physical systems with emergent collective computational abilities, *Proceedings of the National Academy of Sciences*, Vol. 79, pp. 2554-2558, 1982.
7. D.E. Rumelhart and J.L. McClelland, *Parallel Distributed Processing*: *Explorations in the Microstructure of Cognition*. The MIT Press, 1986.
8. B.A. Pearlmutter, Learning state space trajectories in recurrent neural networks, *Neural Computation*, Vol. 1, No. 2, pp. 263-269, 1989.
9. B.W. Mel, *Connectionist Robot Motion Planning*. San Diego, CA: Academic Press, 1990.

CHAPTER 11

Perceptron and Adaline

11.1 INTRODUCTION

This chapter describes single layer neural networks, including some of the classical approaches to the neural computing and learning problem. In the first part of this chapter we discuss the representational power of the single layer networks and their learning algorithms and will give some examples of using the networks. In the second part we will discuss the representational limitations of single layer networks.

Two 'classical' models will be described in the first part of the chapter: the Perceptron, proposed by Rosenblatt and the Adaline, presented by Widrow and Hoff.

11.2 NETWORKS WITH THRESHOLD ACTIVATION FUNCTIONS

A single layer feed-forward network consists of one or more output neurons o, each of which is connected with a weighting factor w_{io} to all of the inputs i. In the simplest case the network has only two inputs and a single output, as sketched in Fig. 11.1 (we leave the output index o out). The input of the neuron is the weighted sum of the inputs plus the bias term. The output of the network is formed by the activation of the output neuron, which is some function of the input:

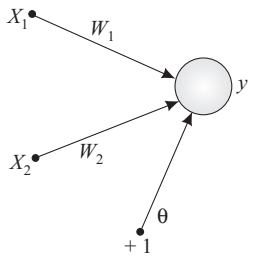

Fig. 11.1 Single layer network with one output and two inputs.

$$y = F\left(\sum_{i=1}^{2} w_i x_i + \theta\right) \qquad ...(11.1)$$

The activation function F can be linear so that we have a linear network, or non-linear. In this section we consider the threshold (sgn) function:

$$F(s) = \begin{cases} +1 & \text{if } s > 0 \\ -1 & \text{otherwise} \end{cases} \qquad ...(11.2)$$

The output of the network thus is either +1 or –1, depending on the input. The network can now be used for a classification task: it can decide whether an input pattern belongs to one of two classes. If the total input is positive, the pattern will be assigned to class +1, if the total input is negative, the sample will be assigned to class +1. The separation between the two classes in this case is a straight line, given by the equation:

$$w_1 x_1 + w_2 x_2 + \theta = 0 \qquad ...(11.3)$$

The single layer network represents a linear discriminant function.

A geometrical representation of the linear threshold neural network is given in Fig. 11.2. Equation (11.3) can be written as

$$x_2 = -\frac{w_1}{w_2} x_1 - \frac{\theta}{w_2} \qquad ...(11.4)$$

and we see that the weights determine the slope of the line and the bias determines the 'offset', i.e. how far the line is from the origin. Note that also the weights can be plotted in the input space: the weight

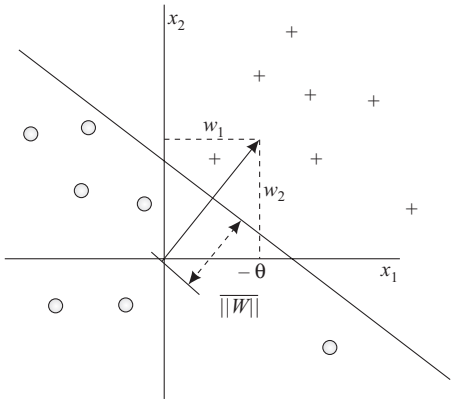

Fig. 11.2 Geometric representation of the discriminant function and the weights.

vector is always perpendicular to the discriminant function.

Now that we have shown the representational power of the single layer network with linear threshold units, we come to the second issue: how do we learn the weights and biases in the network? We will describe two learning methods for these types of networks: the 'perceptron' learning rule and the `delta' or `LMS' rule. Both methods are iterative procedures that adjust the weights. A learning sample is presented to the network. For each weight the new value is computed by adding a correction to the old value. The threshold is updated in a same way:

$$w_i(t+1) = w_i(t) + \Delta w_i(t) \quad ...(11.5)$$

$$\theta(t+1) = \theta(t) + \Delta\theta(t) \quad ...(11.6)$$

The learning problem can now be formulated as: how do we compute $\Delta w_i(t)$ and $\Delta\theta(t)$ in order to classify the larning patterns correctly?

11.3 PERCEPTRON LEARNING RULE AND CONVERGENCE THEOREM

11.3.1 Perceptron Learning Rule

Suppose we have a set of learning samples consisting of an input vector x and a desired output $d(x)$. For a classification task the $d(x)$ is usually +1 or –1. The perceptron learning rule is very simple and can be stated as follows:

1. Start with random weights for the connections;
2. Select an input vector x from the set of training samples;
3. If $y \neq d(x)$ (the perceptron gives an incorrect response), modify all connections w_i according to: $\Delta w_i = d(x)x_i$;
4. Go back to 2.

Note that the procedure is very similar to the Hebb rule; the only difference is that, when the network responds correctly, no connection weights are modified. Besides modifying the weights, we must also modify the threshold θ. This θ is considered as a connection w_0 between the output neuron and a 'dummy' predicate unit which is always on: $x_0 = 1$. Given the perceptron learning rule as stated above, this threshold is modified according to:

$$\Delta\theta = \begin{cases} 0 & \text{if the perceptron responds correctly} \\ d(x) & \text{otherwise} \end{cases} \quad ...(11.7)$$

11.3.2 Convergence Theorem

For the learning rule there exists a convergence theorem, which states the following: ***"If there exists a set of connection weights w* which is able to perform the transformation y = d(x), the perceptron learning rule will converge to some solution (which may or may not be the same as w*) in a finite number of steps for any initial choice of the weights".***

Proof: Given the fact that the length of the vector w^* does not play a role (because of the sgn operation), we take $\|w^*\| = 1$. Because w^* is a correct solution, the value $|w^* \circ x|$, where o denotes dot or inner product, will be greater than 0 or: there exists a $\delta > 0$ such that $|w^* \circ x| > \delta$ for all inputs x.

Now define $\cos \alpha = \dfrac{w \circ w^*}{||w||}$.

When according to the perceptron learning rule, connection weights are modified at a given input x, we know that $\Delta w = d(x)x$, and the weight after modification is $w' = w + \Delta w$. From this it follows that:

$$\begin{aligned} w' \circ w^* &= w \circ w^* + d(x) \circ w^* \circ x \\ &= w \circ w^* + sgn(w^* \circ x)\, w^* \circ x \\ &> w \circ w^* + \delta \\ ||w'||^2 &= ||w + d(x)x||^2 \\ &= w^2 + 2d\,(x)\, w \circ x + x^2 \\ &< w^2 + x^2 \qquad \text{(because } d(x) = -\text{sgn}\,[w \circ x]) \\ &= w^2 + M \end{aligned}$$

After t modifications we have:

$$w(t) \circ w^* > w \circ w^* + t\delta$$
$$||w(t)||^2 < w^2 + tM$$

such that

$$\cos \alpha(t) = \frac{w^* \circ w(t)}{||w(t)||}$$

$$> \frac{w^* \circ w + t\delta}{\sqrt{w^2 + tM}}$$

From this follows that

$$\lim_{t\to\infty} \cos \alpha(t) = \lim_{t\to\infty} \frac{\delta}{\sqrt{M}} \sqrt{t} = \infty \text{ while } \cos \alpha \le 1.$$

The conclusion is that there must be an upper limit t_{max} for t. the system modifies its connections only a limited number of times. In other words, after maximally t_{max} modifications of the weights the perceptron is correctly performing the mapping. t_{max} will be reached when $\cos \alpha = 1$. If we start with connections $w = 0$,

$$t_{max} = \frac{M}{\delta^2} \qquad \text{...(11.8)}$$

Example 11.1: A perceptron is initialized with the following weights: $w_1 = 1$; $w_2 = 2$; $\theta = -2$. The perceptron learning rule is used to learn a correct discriminant function for a number of samples, sketched in Fig. 11.3.

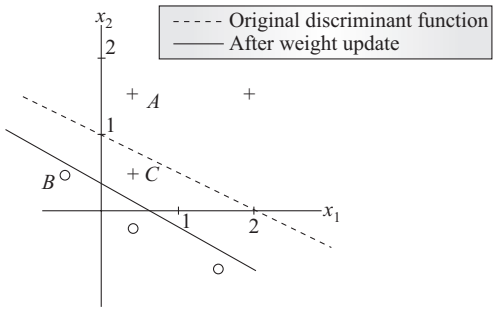

Fig. 11.3 Discriminant function before and after weight update.

The first sample A, with values $x = (0{:}5; 1{:}5)$ and target value $d(x) = +1$ is presented to the network. From equation (11.1) it can be calculated that the network output is +1, so no weights are adjusted. The same is the case for point *B*, with values $x = (–0{:}5; 0{:}5)$ and target value $d(x) = -1$; the network output is negative, so no change. When presenting point *C* with values $x = (0{:}5; 0{:}5)$ the network output will be –1, while the target value $d(x) = +1$.

According to the perceptron learning rule, the weight changes are:

$$\Delta w_1 = 0{:}5, \Delta w_2 = 0{:}5, \theta = 1.$$

The new weights are now:

$$w_1 = 1{:}5, w_2 = 2{:}5, \theta = –1,$$

and sample *C* is classified correctly.

In Fig. 11.3 the discriminant function before and after this weight update is shown.

11.4 ADAPTIVE LINEAR ELEMENT (Adaline)

An important generalisation of the perceptron training algorithm was presented by Widrow and Hoff as the ‘least mean square’ (LMS) learning procedure, also known as the delta rule. The main functional di_erence with the perceptron training rule is the way the output of the system is used in the learning rule. The perceptron learning rule uses the output of the threshold function (either –1 or +1) for learning. The delta-rule uses the net output without further mapping into output values –1 or +1.

The learning rule was applied to the ‘adaptive linear element’, also named Adaline, developed by Widrow and Hoff. In a simple physical implementation (Fig. 11.4) this device consists of a set of controllable resistors connected to a circuit, which can sum up currents caused by the input voltage signals. Usually the central block, the summer, is also followed by a quantiser, which outputs either +1 or –1, depending on the polarity of the sum.

Although the adaptive process is here exemplified in a case when there is only one output, it may be clear that a system with many parallel outputs is directly implementable by multiple units of the above kind.

If the input conductances are denoted by w_i, $i = 0; 1,..., n$, and the input and output signals by x_i and y, respectively, then the output of the central block is defined to be

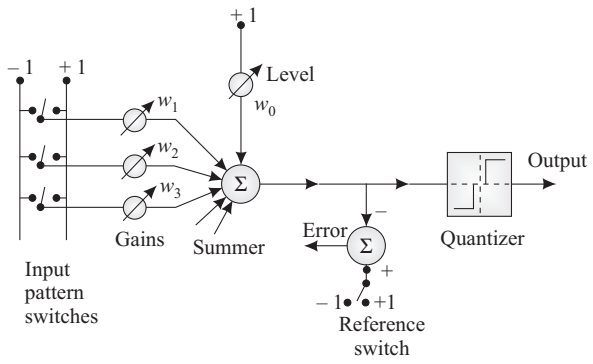

Fig. 11.4 The adaline.

$$y = \sum_{i=1}^{n} w_i x_i + \theta \qquad ...(11.9)$$

where $\theta = w_0$. The purpose of this device is to yield a given value $y = d^p$ at its output when the set of values x_i^p, $i = 1; 2, \ldots, n$, is applied at the inputs. The problem is to determine the coeficients w_i, $i = 0, 1, \ldots, n$, in such a way that the input-output response is correct for a large number of arbitrarily chosen signal sets. If an exact mapping is not possible, the average error must be minimised, for instance, in the sense of least squares. An adaptive operation means that there exists a mechanism by which the w_i can be adjusted, usually iteratively, to attain the correct values. For the Adaline, Widrow introduced the delta rule to adjust the weights.

11.5 THE DELTA RULE

For a single layer network with an output unit with a linear activation function the output is simply given by

$$y = \sum_{j} w_j x_j + \theta \qquad ...(11.10)$$

Such a simple network is able to represent a linear relationship between the value of the output unit and the value of the input units. By thresholding the output value, a classifier can be constructed (such as Adaline), but here we focus on the linear relationship and use the network for a function approximation task. In high dimensional input spaces the network represents a (hyper) plane and it will be clear that also multiple output units may be defined.

Suppose we want to train the network such that a hyperplane is fitted as well as possible to a set of training samples consisting of input values x^p and desired (or target) output values d^p. For every given input sample, the output of the network differs from the target value d^p by $(d^p - y^p)$, where y^p is the actual output for this pattern. The delta-rule now uses a cost-or error-function based on these differences to adjust the weights.

The error function, as indicated by the name least mean square, is the summed squared error. That is, the total error E is defined to be

$$E = \sum_p E^p = \frac{1}{2}\sum_p (d^p - y^p)^2 \qquad ...(11.11)$$

where the index p ranges over the set of input patterns and E^p represents the error on pattern p. The LMS procedure finds the values of all the weights that minimize the error function by a method called gradient descent. The idea is to make a change in the weight proportional to the negative of the derivative of the error as measured on the current pattern with respect to each weight:

$$\Delta_p w_j = -\gamma \frac{\partial E^p}{\partial w_j} \qquad ...(11.12)$$

where γ is a constant of proportionality. The derivative is

$$\frac{\partial E^p}{\partial w_j} = \frac{\partial E^p}{\partial y^p} \frac{\partial y^p}{\partial w_j}. \qquad ...(11.13)$$

Because of the linear units, eq. (11.10),

$$\frac{\partial y^p}{\partial w_j} = x_j \qquad ...(11.14)$$

and

$$\frac{\partial E^p}{\partial y^p} = -(d^p - y^p) \qquad ...(11.15)$$

such that

$$\Delta_p w_j = \gamma \delta^p x_j \qquad ...(11.16)$$

where $\delta^p = d^p - y^p$ is the difference between the target output and the actual output for pattern p.

The delta rule modifies weight appropriately for target and actual outputs of either polarity and for both continuous and binary input and output units. These characteristics have opened up a wealth of new applications.

11.6 EXCLUSIVE-OR PROBLEM

In the previous sections we have discussed two learning algorithms for single layer networks, but we have not discussed the limitations on the representation of these networks.

Table 11.1 Exclusive-or truth table.

x_0	x_1	d
–1	–1	–1
–1	1	1
1	–1	1
1	1	1

One of Minsky and Papert's most discouraging results shows that a single layer perceptron cannot represent a simple exclusive-or function. Table 3.1 shows the desired relationships between inputs and output units for this function.

In a simple network with two inputs and one output, as depicted in Fig. 11.1, the net input is equal to:

$$s = w_1x_1 + w_2x_2 + \theta \qquad ...(11.17)$$

According to eq. (11.1), the output of the perceptron is zero when s is negative and equal to one when s is positive. In Fig. 11.5 a geometrical representation of the input domain is given. For a constant θ, the output of the perceptron is equal to one on one side of the dividing line which is defined by:

$$w_1x_1 + w_2x_2 = -\theta \qquad ...(11.18)$$

and equal to zero on the other side of this line.

To see that such a solution cannot be found, take a loot at Fig. 11.5. The input space consists of four points, and the two solid circles at (1, –1) and (–1, 1) cannot be separated by a straight line from the two open circles at (–1, –1) and (1, 1). The obvious question to ask is: How can this problem be overcome? Minsky and Papert prove that for binary inputs, any transformation can be carried out by adding a layer of predicates which are connected to all inputs. The proof is given in the next section.

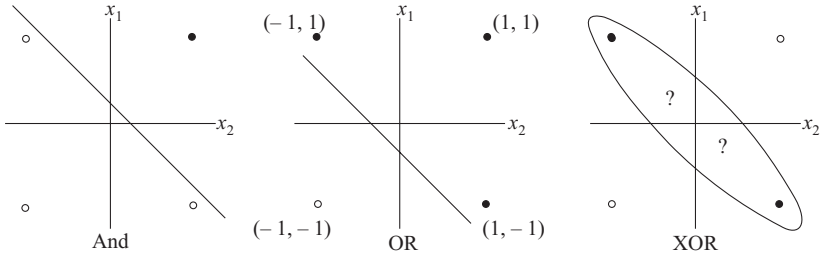

Fig. 11.5 Geometric representation of input space

For the specific *XOR* problem we geometrically show that by introducing hidden units, thereby extending the network to a multi-layer perceptron, the problem can be solved. Fig. 11.6a demonstrates that the four input points are now embedded in a three-dimensional space defined by the two inputs plus the single hidden unit. These four points are now easily separated by a linear manifold (plane) into two groups, as desired. This simple example demonstrates that adding hidden units increases the class of

problems that are soluble by feed-forward, perceptron- like networks. However, by this generalization of the basic architecture we have also incurred a serious loss: we no longer have a learning rule to determine the optimal weights.

(a) The perceptron of Fig. 11.1 with an extra hidden unit. With the indicated values of the weights w_{ij} (next to the connecting lines) and the thresholds θ_i (in the circles) this perceptron solves the *XOR* problem. (b) This is accomplished by mapping the four points of Fig. 11.6 onto the four points indicated here; clearly, separation (by a linear manifold) into the required groups is now possible.

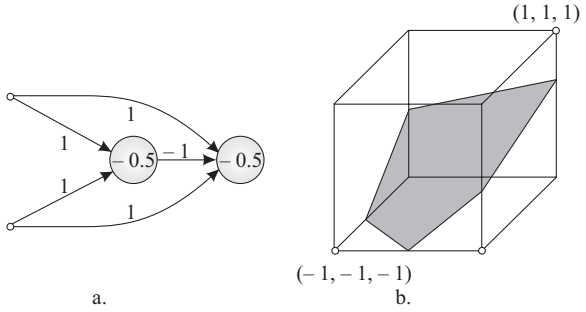

Fig. 11.7 Solution of the XOR problem.

11.7 MULTI-LAYER PERCEPTRONS CAN DO EVERYTHING

In the previous section we showed that by adding an extra hidden unit, the *XOR* problem can be solved. For binary units, one can prove that this architecture is able to perform any transformation given the correct connections and weights. The most primitive is the next one. For a given transformation $y = d(x)$, we can divide the set of all possible input vectors into two classes:

$$X^+ = \{x|d(x) = 1\} \text{ and } X^- = \{x|d(x) - 1\} \qquad ...(11.19)$$

Since there are N input units, the total number of possible input vectors x is $2N$. For every $x^p \in X^+$ a hidden unit h can be reserved of which the activation y_h is 1 if and only if the specific pattern p is present at the input: we can choose its weights w_{ih} equal to the specific pattern x^p and the bias θ_h equal to 1 - N such that

$$y_h^p = \text{sgn}\left(\sum_i w_{ih} x_i^p - N + \frac{1}{2}\right) \qquad ...(11.20)$$

is equal to 1 for $x^p = w_h$ only. Similarly, the weights to the output neuron can be chosen such that the output is one as soon as one of the M predicate neurons is one:

$$y_o^p = \text{sgn}\left(\sum_{h=1}^{M} y_h + M - \frac{1}{2}\right) \qquad ...(11.21)$$

This perceptron will give $y_0 = 1$ only if $x \in X^+$: it performs the desired mapping. The problem is the large number of predicate units, which is equal to the number of patterns in X^+, which is maximally 2^N. Of course we can do the same trick for X^-, and we will always take the minimal number of mask units, which is maximally 2^{N-1}. A more elegant proof is given by Minsky and Papert, but the point is that for complex transformations the number of required units in the hidden layer is exponential in N.

QUESTION BANK

1. Explain single layer neural network with one output and two inputs.
2. Describe the perceptron learning rule.
3. Derive the convergence theorem for perceptron learning rule.
4. Explain Adaline neural network.
5. Explain the delta rule used to adjust the weights of Adaline network.
6. Single layer perceptron cannot represent exclusive-OR. Justify this statement.
7. What are the advantages of multiplayer perceptron over single layer perceptron?

REFERENCES

1. F. Rosenblatt, *Principles of Neurodynamics*, New York: Spartan Books, 1959.
2. B. Widrow, and M.E. Hoff, *Adaptive Switching Circuits*, In 1960 Ire Wescon Convention Record, Dunno, 1960.
3. D.O. Hebb, *The Organization of Behaviour*. New York: Wiley. 1949.
4. M. Minsky, and S. Papert, *Perceptrons*: *An Introduction to Computational Geometry*, The MIT Press, 1969.

CHAPTER 12

Back-Propagation

12. 1 INTRODUCTION

As we have seen in the previous chapter, a single-layer network has severe restrictions: the class of tasks that can be accomplished is very limited. In this chapter we will focus on feed forward networks with layers of processing units.

Minsky and Papert showed in 1969 that a two layer feed-forward network can overcome many restrictions, but did not present a solution to the problem of how to adjust the weights from input to hidden units. An answer to this question was presented by Rumelhart, Hinton and Williams in 1986, and similar solutions appeared to have been published earlier (Parker, 1985; Cun, 1985).

The central idea behind this solution is that the errors for the units of the hidden layer are determined by back-propagating the errors of the units of the output layer. For this reason the method is often called the back-propagation learning rule. Back-propagation can also be considered as a generalization of the delta rule for non-linear activation functions and multilayer networks.

12.2 MULTI - LAYER FEED - FORWARD NETWORKS

A feed-forward network has a layered structure. Each layer consists of units, which receive their input from units from a layer directly below and send their output to units in a layer directly above the unit. There are no connections within a layer. The N_i inputs are fed into the first layer of $N_{h,\,1}$ hidden units. The input units are merely 'fan-out' units; no processing takes place in these units. The activation of a hidden unit is a function F_i of the weighted inputs plus a bias, as given in eq. (10.4). The output of the hidden units is distributed over the next layer of $N_{h,\,2}$ hidden units, until the last layer of hidden units, of which the outputs are fed into a layer of No output units (see Fig. 12.1).

Although back-propagation can be applied to networks with any number of layers, just as for networks with binary units (section 11.7) it has been shown (Cybenko, 1989; Funahashi, 1989; Hornik, Stinchcombe, & White, 1989; Hartman, Keeler, & Kowalski, 1990) that only one layer of hidden units suffices to approximate any function with finitely many discontinuities to arbitrary precision, provided the activation functions of the hidden units are non-linear (the universal approximation theorem).

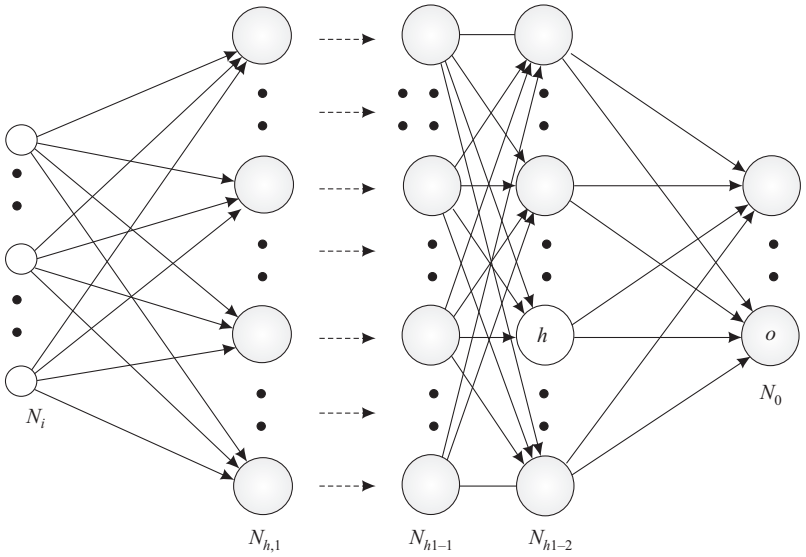

Fig. 12.1 A multi-layer network with layers of units.

In most applications a feed-forward network with a single layer of hidden units is used with a sigmoid activation function for the units.

12.3 THE GENERALISED DELTA RULE

Since we are now using units with nonlinear activation functions, we have to generalise the delta rule, which was presented in chapter 11 for linear functions to the set of non-linear activation functions. The activation is a differentiable function of the total input, given by

$$y_k^p = F(S_k^p) \qquad \text{... (12.1)}$$

in which

$$s_k^p = \sum_j w_{jk} y_k^p + \theta_k \qquad \text{...(12.2)}$$

To get the correct generalization of the delta rule as presented in the previous chapter, we must set

$$\Delta_p w_{jk} = -\gamma \frac{\partial E^p}{\partial w_{jk}} \qquad \text{...(12.3)}$$

The error E^p is defined as the total quadratic error for pattern p at the output units:

$$E^p = \frac{1}{2} \sum_{o=1}^{N_o} \left(d_o^p - y_o^p \right)^2 \qquad \text{...(12.4)}$$

where d_o^p is the desired output for unit 0 when pattern p is clamped. We further set $E = \sum_p E^p$ as the summed squared error. We can write

$$\frac{\partial E^p}{\partial w_{jk}} = \frac{\partial E^p}{\partial S_k^p} \frac{\partial S_k^p}{\partial w_{jk}} \quad ...(12.5)$$

By equation (12.2) we see that the second factor is

$$\frac{\partial S_k^p}{\partial w_{jk}} = y_j^p \quad ...(12.6)$$

When we define

$$\delta_k^p = \frac{\partial E^p}{\partial S_k^p} \quad ...(12.7)$$

we will get an update rule which is equivalent to the delta rule as described in the previous chapter, resulting in a gradient descent on the error surface if we make the weight changes according to:

$$\Delta_p w_{jk} = \gamma \delta_k^p y_j^p \quad ...(12.8)$$

The trick is to figure out what δ_k^p should be for each unit k in the network. The interesting result, which we now derive, is that there is a simple recursive computation of these δ's which can be implemented by propagating error signals backward through the network.

To compute δ_k^p we apply the chain rule to write this partial derivative as the product of two factors, one factor reflecting the change in error as a function of the output of the unit and one reflecting the change in the output as a function of changes in the input. Thus, we have

$$\delta_k^p = \frac{\partial E^p}{\partial S_k^p} = \frac{\partial E^p}{\partial y_k^p} \frac{\partial y_k^p}{\partial S_k^p} \quad ...(12.9)$$

Let us compute the second factor. By equation (12.1) we see that

$$\frac{\partial y_k^p}{\partial S_k^p} = F(S_k^p) \quad ...(12.10)$$

which is simply the derivative of the squashing function F for the kth unit, evaluated at the net input S_k^p to that unit. To compute the first factor of equation (12.9), we consider two cases. First, assume that unit k is an output unit $k = o$ of the network. In this case, it follows from the definition of E^p that

$$\frac{\partial E^p}{\partial y_o^p} = -(d_o^p - y_o^p) \quad ... (12.11)$$

which is the same result as we obtained with the standard delta rule. Substituting this and equation (12.10) in equation (12.9), we get

$$\delta_o^p = (d_o^p - y_o^p)\ F_o^{'}\ (S_o^p) \qquad ...(12.12)$$

for any output unit o. Secondly, if k is not an output unit but a hidden unit $k = h$, we do not readily know the contribution of the unit to the output error of the network. However, the error measure can be written as a function of the net inputs from hidden to output layer $E^p = E^p\ (s_1^p, s_2^p, ..., s_j^p, ...)$ and we use the chain rule to write

$$\frac{\partial E^p}{\partial y_h^p} = \sum_{o=1}^{N_o} \frac{\partial E^p}{\partial S_o^p} \frac{\partial S_o^p}{\partial S_h^p} = \sum_{o=1}^{N_o} \frac{\partial E^p}{\partial S_o^p} \frac{\partial}{\partial y_h^p} \sum_{j=1}^{N_o} w_{ko} y_j^p = \sum_{j=1}^{N_o} \frac{\partial E^p}{\partial S_o^p} w_{ho} = \sum_{j=1}^{N_o} \delta_o^p w_{ho} \qquad ... (12.13)$$

Substituting this in equation (12.9) yields

$$\delta_h^p = F(S_h^p) \sum_{j=1}^{N_o} \delta_o^p w_{ho} \qquad ... (12.14)$$

Equations (12.12) and (12.14) give a recursive procedure for computing the δ'*s* for all units in the network, which are then used to compute the weight changes according to equation (12.8). This procedure constitutes the generalized delta rule for a feed-forward network of non-linear units.

12.3.1 Understanding Back-Propagation

The equations derived in the previous section may be mathematically correct, but what do they actually mean? Is there a way of understanding back-propagation other than reciting the necessary equations?

The answer is, of course, yes. In fact, the whole back-propagation process is intuitively very clear. What happens in the above equations is the following. When a learning pattern is clamped, the activation values are propagated to the output units, and the actual network output is compared with the desired output values, we usually end up with an error in each of the output units. Let's call this error e_o for a particular output unit o. We have to bring e_o to zero.

The simplest method to do this is the greedy method: we strive to change the connections in the neural network in such a way that, next time around, the error e_o will be zero for this particular pattern. We know from the delta rule that, in order to reduce an error, we have to adapt its incoming weights according to

$$\Delta w_{ho} = (d_\circ - y_\circ)\ y_h \qquad ...(12.15)$$

That is step one. But it alone is not enough: when we only apply this rule, the weights from input to hidden units are never changed, and we do not have the full representational power of the feed-forward network as promised by the universal approximation theorem.

In order to adapt the weights from input to hidden units, we again want to apply the delta rule. In this case, however, we do not have a value for δ for the hidden units. This is solved by the chain rule which does the following: distribute the error of an output unit o to all the hidden units that is it connected to, weighted by this connection. Differently put, a hidden unit h receives a delta from each output unit o equal to the delta of that output unit weighted with (= multiplied by) the weight of the

connection between those units. In symbols: $\delta_h = \sum_0 \delta_0 w_{ho}$ Well, not exactly: we forgot the activation function of the hidden unit; F' has to be applied to the delta, before the back-propagation process can continue.

12.4 WORKING WITH BACK-PROPAGATION

The application of the generalised delta rule thus involves two phases: During the first phase the input x is presented and propagated forward through the network to compute the output values y_o^p for each output unit. This output is compared with its desired value d_o, resulting in an error signal δ_o^p for each output unit.

The second phase involves a backward pass through the network during which the error signal is passed to each unit in the network and appropriate weight changes are calculated.

12.4.1 Weight Adjustments with Sigmoid Activation Function

The results from the previous section can be summarised in three equations:

- The weight of a connection is adjusted by an amount proportional to the product of an error signal δ, on the unit k receiving the input and the output of the unit j sending this signal along the connection:

$$\Delta_p w_{kj} = \gamma \delta_k^p y_j^p \qquad ...(12.16)$$

- If the unit is an output unit, the error signal is given by

$$\delta_o^p = (d_o^p - y_o^p)\ F_o'(S_o^p) \qquad ...(12.17)$$

Take as the activation function F the 'sigmoid' function as defined in chapter 2:

$$y^p = F(S^p) = \frac{1}{1+e^{-s^p}} \qquad ...(12.18)$$

In this case the derivative is equal to

$$F'(S^p) = \frac{\partial}{\partial S^p}\ \frac{1}{1+e^{-s^p}} = \frac{1}{\left(1+e^{-s^e}\right)^2}\left(-e^{-s^p}\right) = -\frac{1}{\left(1+e^{-s^p}\right)^2}\ \frac{\left(-e^{-s^p}\right)}{\left(1+e^{-s^p}\right)} = y^p(1-y^p) \qquad ...(12.19)$$

such that the error signal for an output unit can be written as:

$$\delta_o^p = (d_o^p - y_o^p)\ y_{o\,o}^p\,(1-y_o^p) \qquad ...(12.20)$$

- The error signal for a hidden unit is determined recursively in terms of error signals of the units to which it directly connects and the weights of those connections. For the sigmoid activation function:

$$\delta_h^p = F'(S_h^p) \sum_{j=1}^{N_o} d_o^p w_{ho} = y_h^p(1 - y_h^p) \sum_{j=1}^{N_o} d_o^p w_{ho} \quad ...(12.21)$$

12.4.2 Learning Rate And Momentum

The learning procedure requires that the change in weight is proportional to $\frac{\partial E^p}{\partial w}$. True gradient descent requires that infinitesimal steps are taken. The constant of proportionality is the learning rate γ. For practical purposes we choose a learning rate that is as large as possible without leading to oscillation. One way to avoid oscillation at large, is to make the change in weight dependent of the past weight change by adding a momentum term:

$$\Delta w_{jk}(t + 1) = \gamma \delta_k^p y_j^p + \alpha \Delta w_{jk}(t) \quad ...(12.22)$$

where t indexes the presentation number and α is a constant which determines the effect of the previous weight change.

The role of the momentum term is shown in Fig. 12.2. When no momentum term is used, it takes a long time before the minimum has been reached with a low learning rate, whereas for high learning rates the minimum is never reached because of the oscillations. When adding the momentum term, the minimum will be reached faster.

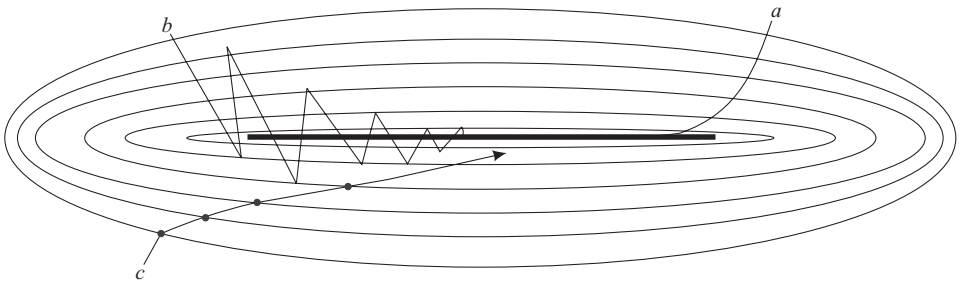

Fig. 12.2 The descent in weight space. (a) for small learning rate; (b) for large learning rate: note the oscillations, and (c) with large learning rate and momentum term added.

12.4.3 Learning Per Pattern

Although, theoretically, the back-propagation algorithm performs gradient descent on the total error only if the weights are adjusted after the full set of learning patterns has been presented, more often than not the learning rule is applied to each pattern separately, i.e., a pattern p is applied, E^p is calculated, and the weights are adapted (p = 1, 2, …, P). There exists empirical indication that this results in faster convergence. Care has to be taken, however, with the order in which the patterns are taught. For example, when using the same sequence over and over again the network may become focused on the first few patterns. This problem can be overcome by using a permuted training method.

Example 12.1: A feed-forward network can be used to approximate a function from examples. Suppose we have a system (for example a chemical process or a financial market) of which we want to know the characteristics. The input of the system is given by the two-dimensional vector x and the output is given by the one-dimensional vector d. We want to estimate the relationship $d = f(x)$ from 80 examples $\{x^p, d^p\}$ as depicted in Fig. 12.3 (top left). A feed-forward network was programmed with two inputs, 10 hidden units with sigmoid activation function and an output unit with a linear activation function. Check for yourself how equation (4.20) should be adapted for the linear instead of sigmoid activation function. The network weights are initialized to small values and the network is trained for 5,000 learning iterations with the back-propagation training rule, described in the previous section. The relationship between x and d as represented by the network is shown in Fig. 12.3 (top right), while the function which generated the learning samples is given in Fig. 12.3 (bottom left). The approximation error is depicted in Fig. 12.3 (bottom right). We see that the error is higher at the edges of the region within which the learning samples were generated. The network is considerably better at interpolation than extrapolation.

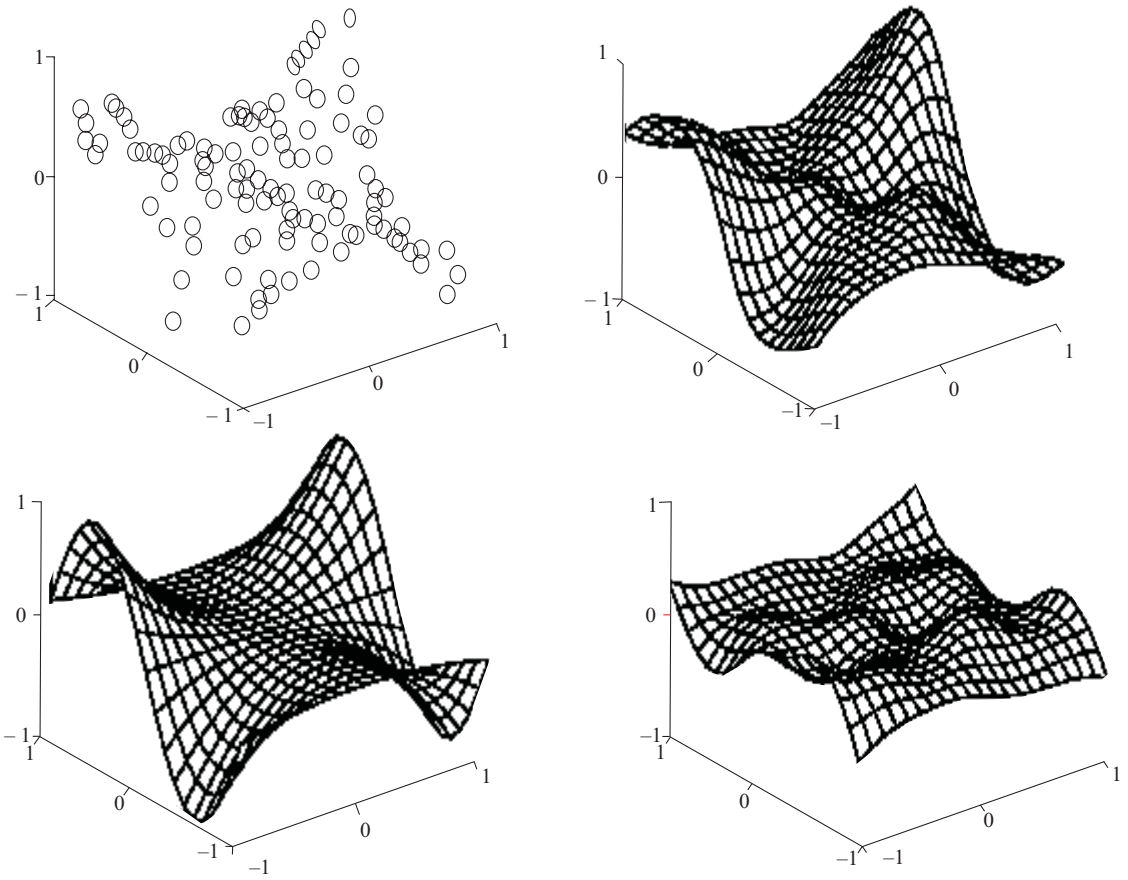

Fig. 12.3 Example of function approximation with a feed forward network. Top left: The original learning samples; Top right: The approximation with the network; Bottom left: The function which generated the learning samples; Bottom right: The error in the approximation.

12.5 OTHER ACTIVATION FUNCTIONS

Although sigmoid functions are quite often used as activation functions, other functions can be used as well. In some cases this leads to a formula, which is known from traditional function approximation theories.

For example, from Fourier analysis it is known that any periodic function can be written as a infinite sum of sine and cosine terms (Fourier series):

$$f(x) = \sum_{n=0}^{\infty} (a_n \cos nx + b_n \sin nx) \qquad ...(12.23)$$

We can rewrite this as a summation of sine terms

$$f(x) = a_0 + \sum_{n=1}^{\infty} c_n \sin (nx + \theta_n) \qquad ...(12.24)$$

with $c_n = \sqrt{a_n^2 + b_n^2}$ and $\theta_n = \arctan (b/a)$. This can be seen as a feed-forward network with a single input unit for x; a single output unit for $f(x)$ and hidden units with an activation function $F = \sin (s)$. The factor a_0 corresponds with the bias of the output unit, the factors c_n correspond with the weighs from hidden to output unit; the phase factor θ_n corresponds with the bias term of the hidden units and the factor n corresponds with the weights between the input and hidden layer.

The basic difference between the Fourier approach and the back-propagation approach is that the in the Fourier approach the 'weights' between the input and the hidden units (these are the factors n) are fixed integer numbers which are analytically determined, whereas in the back-propagation approach these weights can take any value and are typically learning using a learning heuristic.

To illustrate the use of other activation functions we have trained a feed-forward network with one output unit, four hidden units, and one input with ten patterns drawn from the function $f(x) = \sin(2x) \sin(x)$. The result is depicted in Fig. 12.4. The same function (albeit with other learning points) is learned with a network with eight sigmoid hidden units (see Figure 12.5). From the figures it is clear that it pays off to use as much knowledge of the problem at hand as possible.

12.6 DEFICIENCIES OF BACK-PROPAGATION

Despite the apparent success of the back-propagation learning algorithm, there are some aspects, which make the algorithm not guaranteed to be universally useful. Most troublesome is the long training process. This can be a result of a non-optimum learning rate and momentum. A lot of advanced algorithms based on back-propagation learning have some optimized method to adapt this learning rate, as will be discussed in the next section. Outright training failures generally arise from two sources: network paralysis and local minima.

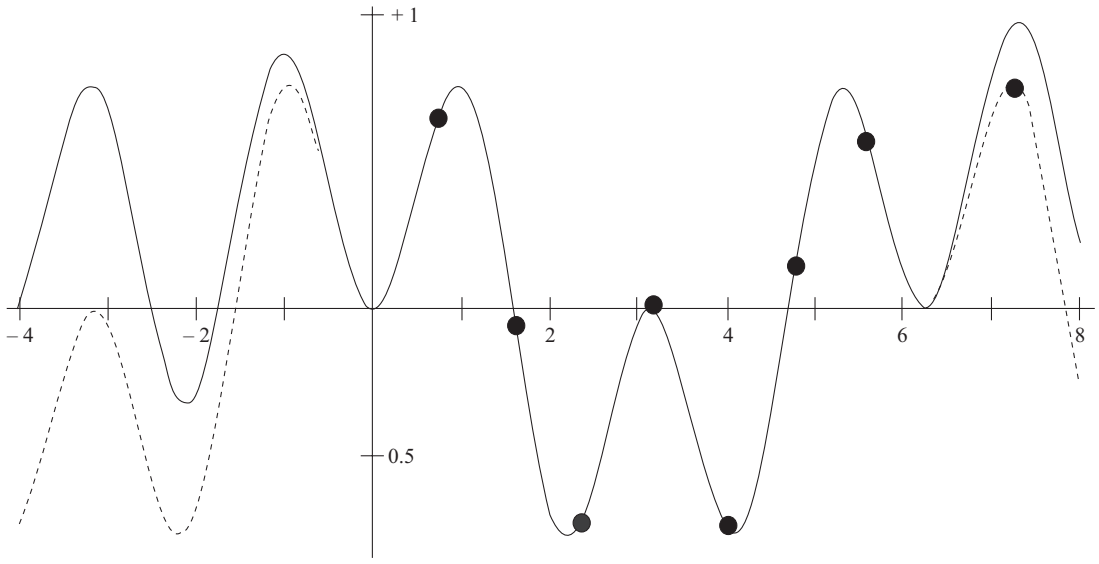

Fig. 12.4 The periodic function $f(x) = \sin(2x) \sin(x)$ approximated with sine activation functions.

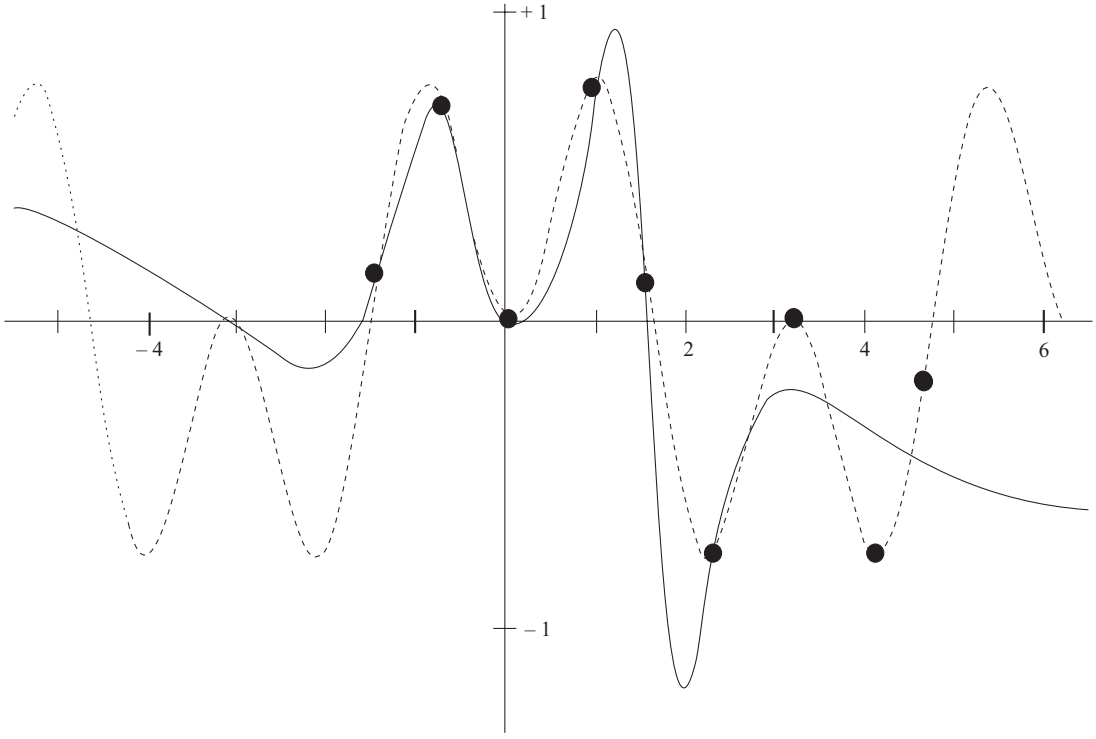

Fig. 12.5 The periodic function $f(x) = \sin(2x) \sin(x)$ approximated with sigmoid activation functions.

12.6.1 Network Paralysis

As the network trains, the weights can be adjusted to very large values. The total input of a hidden unit or output unit can therefore reach very high (either positive or negative) values, and because of the sigmoid activation function the unit will have an activation very close to zero or very close to one. As is clear from equations (12.20) and (12.21), the weight adjustments which are proportional to $y_k^p(1-y_k^p)$ will be close to zero, and the training process can come to a virtual standstill.

12.6.2 Local Minima

The error surface of a complex network is full of hills and valleys. Because of the gradient descent, the network can get trapped in a local minimum when there is a much deeper minimum nearby. Probabilistic methods can help to avoid this trap, but they tend to be slow. Another suggested possibility is to increase the number of hidden units. Although this will work because of the higher dimensionality of the error space, and the chance to get trapped is smaller, it appears that there is some upper limit of the number of hidden units which, when exceeded, again results in the system being trapped in local minima.

12.7 ADVANCED ALGORITHMS

Many researchers have devised improvements of and extensions to the basic back-propagation algorithm described above. It is too early for a full evaluation: some of these techniques may prove to be fundamental, others may simply fade away. A few methods are discussed in this section.

May be the most obvious improvement is to replace the rather primitive steepest descent method with a direction set minimization method, e.g., conjugate gradient minimization. Note that minimization along a direction u brings the function f at a place where its gradient is perpendicular to u (otherwise minimization along u is not complete).

Instead of following the gradient at every step, a set of n directions is constructed which are all conjugate to each other such that minimization along one of these directions u_j does not spoil the minimization along one of the earlier directions u_i, i.e., the directions are non-interfering. Thus one minimization in the direction of u_i suffices, such that n minimizations in a system with n degrees of freedom bring this system to a minimum (provided the system is quadratic). This is different from gradient descent, which directly minimizes in the direction of the steepest descent (Press, Flannery, Teukolsky, & Vetterling, 1986).

Suppose the function to be minimized is approximated by its Taylor series

$$f(x) = f(p) + \sum_i \left.\frac{\partial f}{\partial x_i}\right|_p x_i + \frac{1}{2}\sum_{i,j} \left.\frac{\partial^2 f}{\partial x_i \partial x_j}\right|_p x_i x_j + \ldots \approx \frac{1}{2} x^T A x - b^T x + c \qquad \ldots(12.25)$$

where T denotes transpose, and

$$c \equiv f(p)$$

$$b \equiv -\nabla f\big|_p$$

$$[A]_{ij} = \left.\frac{\partial^2 f}{\partial x_i \partial x_j}\right|_p \qquad ...(12.26)$$

A is a symmetric positive definite $n \times n$ matrix, the Hessian of f at p. The gradient of f is

$$\nabla f = A\mathrm{x} - b \qquad ...(12.27)$$

such that a change of x results in a change of the gradient as

$$\delta(\nabla f) = A(\delta x) \qquad ...(12.28)$$

Now suppose f was minimized along a direction ui to a point where the gradient $-g_{i+1}$ of f is perpendicular to u_i, i.e.,

$$u_i^T \mathrm{g}_{i+1} = 0 \qquad ...(12.29)$$

and a new direction u_{i+1} is sought. In order to make sure that moving along u_{i+1} does not spoil minimization along u_i we require that the gradient of f remain perpendicular to u_i, i.e.,

$$u_i^T g_{i+2} = 0 \qquad ...(12.30)$$

otherwise we would once more have to minimise in a direction which has a component of u_i.

Combining (12.29) and (12.30), we get

$$0 = u_i^T(g_{i+1} - g_{i+2}) = u_i^T \delta(\nabla f) = u_i^T A u_{i+1} \qquad ...(12.31)$$

When eq. (12.31) holds for two vectors u_i and u_{i+1} they are said to be conjugate.

Now, starting at some point p_0, the first minimization direction u_0 is taken equal to $g_0 = -\nabla f(p_0)$, resulting in a new point p_1. For $i \geq 0$, calculate the directions

$$u_{i+1} = \mathrm{g}_{i+1} + \gamma_i u_i \qquad ...(12.32)$$

where γ_i is chosen to make $u_i^T A u_{i-1}$ and the successive gradients perpendicular, i.e.,

$$\gamma_i = \frac{g_{i+1}^T g_{i+1}}{g_i^T g_i} \text{ with } g_k = -\nabla f|_{pk} \text{ for all } k \geq 0 \qquad ...(12.33)$$

Next, calculate $p_{i+2} = p_{i+1} + \lambda_{i+1} u_{i+1}$ where λ_{i+1} is chosen so as to minimize $f(P_{i+2})$[3].

It can be shown that the u's thus constructed are all mutually conjugate (e.g., see (Stoer & Bulirsch, 1980)). The process described above is known as the Fletcher-Reeves method, but there are many variants, which work more or less the same (Hestenes & Stiefel, 1952; Polak, 1971; Powell, 1977).

Although only n iterations are needed for a quadratic system with n degrees of freedom, due to the fact that we are not minimizing quadratic systems, as well as a result of round-off errors, the n directions have to be followed several times (see Fig. 12.6). Powell introduced some improvements to correct for behaviour in non-quadratic systems. The resulting cost is $O(n)$ which is significantly better than the linear convergence 4 of steepest descent.

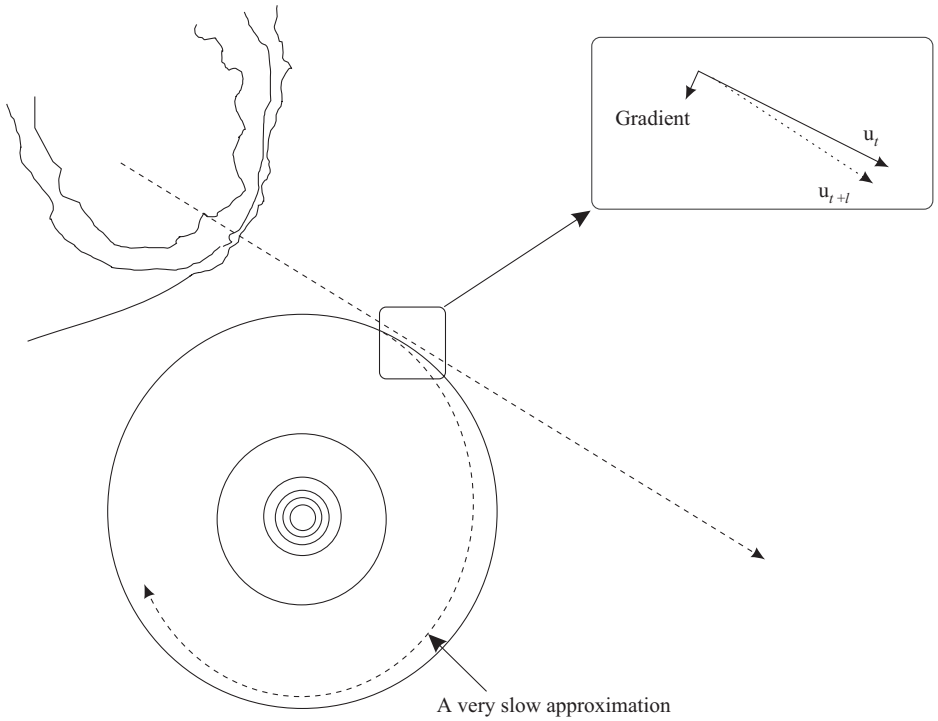

Fig. 12.6 Slow decrease with conjugate gradient in non-quadratic systems. [The hills on the left are very steep, resulting in a large search vector u_i. When the quadratic portion is entered the new search direction is constructed from the previous direction and the gradient, resulting in a spiraling minimization. This problem can be overcome by detecting such spiraling minimizations and restarting the algorithm with $u_0 = -\nabla f$].

Some improvements on back-propagation have been presented based on an independent adaptive arning rate parameter for each weight.

Van den Boomgaard and Smeulders (Boomgaard & Smeulders, 1989) show that for a feed-forward network without hidden units an incremental procedure to find the optimal weight matrix W needs an adjustment of the weights with

$$\Delta w(t+1) = \gamma(t+1)\,[d(t+1) - w(t) \times (t+1)] \times (t+1) \qquad ...(12.34)$$

in which γ is not a constant but an variable $(N_i + 1) \times (N_i + 1)$ matrix which depends on the input vector. By using a priori knowledge about the input signal, the storage requirements for can be reduced.

Silva and Almeida (Silva & Almeida, 1990) also show the advantages of an independent step size for each weight in the network. In their algorithm the learning rate is adapted after every learning pattern:

$$\gamma_{jk}(t+1) = \begin{cases} u\gamma_{jk}(t) & \text{if } \dfrac{\partial E(t+1)}{\partial w_{jk}} \text{ and } \dfrac{\partial E(t)}{\partial w_{jk}} \text{ have the same signs} \\ d\gamma_{jk}(t) & \text{if } \dfrac{\partial E(t+1)}{\partial w_{jk}} \text{ and } \dfrac{\partial E(t)}{\partial w_{jk}} \text{ have the opposite signs} \end{cases} \qquad ...(12.35)$$

where u and d are positive constants with values slightly above and below unity, respectively. The idea is to decrease the learning rate in case of oscillations.

12.8 HOW GOOD ARE MULTI-LAYER FEED-FORWARD NETWORKS?

From the example shown in Fig. 12.3 is clear that the approximation of the network is not perfect. The resulting approximation error is influenced by:

1. The learning algorithm and number of iterations. This determines how good the error on the training set is minimized.
2. The number of learning samples. This determines how good the training samples represent the actual function.
3. The number of hidden units. This determines the 'expressive power' of the network. For 'smooth' functions only a few number of hidden units are needed, for wildly fluctuating functions more hidden units will be needed.

In the previous sections we discussed the learning rules such as back-propagation and the other gradient based learning algorithms, and the problem of finding the minimum error. In this section we particularly address the effect of the number of learning samples and the effect of the number of hidden units.

We first have to define an adequate error measure. All neural network training algorithms try to minimize the error of the set of learning samples which are available for training the network. The average error per learning sample is defined as the learning error rate error rate:

$$E_{\text{learning}} = \frac{1}{P_{\text{learning}}} \sum_{p=1}^{P_{\text{learning}}} E^p \qquad ...(12.36)$$

in which E^p is the difference between the desired output value and the actual network output for the learning samples:

$$E^p = \frac{1}{2} \sum_{0=1}^{N_o} (d_o^p - y_o^p) \qquad ...(12.37)$$

This is the error, which is measurable during the training process.

It is obvious that the actual error of the network will differ from the error at the locations of the training samples. The difference between the desired output value and the actual network output should be integrated over the entire input domain to give a more realistic error measure. This integral can be estimated if we have a large set of samples.

We now define the test error rate as the average error of the test set:

$$E_{\text{test}} = \frac{1}{P_{\text{test}}} \sum_{p=1}^{P_{\text{test}}} E^p \qquad ...(12.38)$$

In the following subsections we will see how these error measures depend on learning set size and number of hidden units.

12.8.1 The Effect Of the Number of Learning Samples

A simple problem is used as example: a function $y = f(x)$ has to be approximated with a feed-forward neural network. A neural network is created with an input, 5 hidden units with sigmoid activation function and a linear output unit. Suppose we have only a small number of learning samples (e.g., 4) and the networks is trained with these samples. Training is stopped when the error does not decrease anymore.

The original (desired) function is shown in Fig. 4.7A as a dashed line. The learning samples and the approximation of the network are shown in the same figure. We see that in this case E_{learning} is small (the network output goes perfectly through the learning samples) but E_{test} is large: the test error of the network is large. The approximation obtained from 20 learning samples is shown in Fig. 12.7B. The E_{learning} is larger than in the case of 5 learning samples, but the E_{test} is smaller.

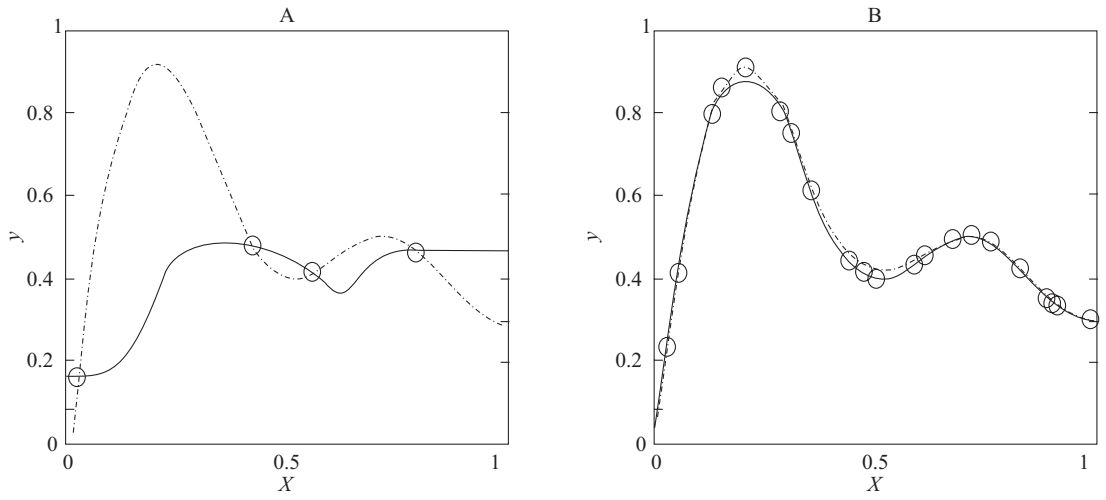

Fig. 12.7 Effect of the learning set size on the generalization. The dashed line gives the desired function, the learning samples are depicted as circles and the approximation by the network is shown by the drawn line. 5 hidden units are used. a) 4 learning samples. b) 20 learning samples.

This experiment was carried out with other learning set sizes, where for each learning set size the experiment was repeated 10 times. The average learning and test error rates as a function of the learning set size are given in Fig. 12.8. Note that the learning error increases with an increasing learning set size, and the test error decreases with increasing learning set size.

A low learning error on the (small) learning set is no guarantee for a good network performance! With increasing number of learning samples the two error rates converge to the same value. This value depends on the representational power of the network: given the optimal weights, how good is the approximation. This error depends on the number of hidden units and the activation function. If the learning error rate does not converge to the test error rate the learning procedure has not found a global minimum.

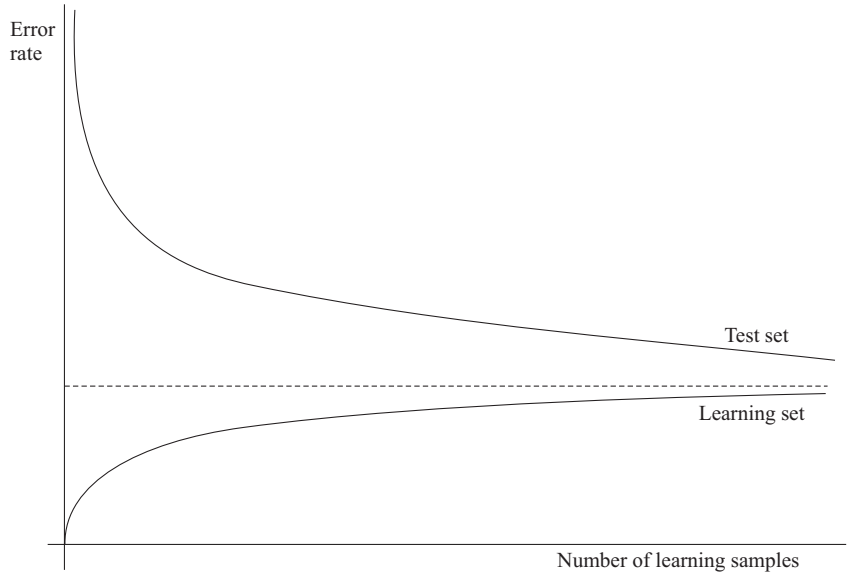

Fig. 12.8 Effect of the learning set size on the error rate. The average error rate and the average test error rate are as a function of the number of learning samples.

12.8.2 The Effect of the Number of Hidden Units

The same function as in the previous subsection is used, but now the number of hidden units is varied. The original (desired) function, learning samples and network approximation is shown in Fig. 4.9A for 5 hidden units and in Fig. 4.9B for 20 hidden units. The effect visible in Fig. 4.9B is called over training. The network fits exactly with the learning samples, but because of the large number of hidden units the function which is actually represented by the network is far more wild than the original one. Particularly in case of learning samples which contain a certain amount of noise (which all real-world data have), the network will 'fit the noise' of the learning samples instead of making a smooth approximation.

This example shows that a large number of hidden units leads to a small error on the training set but not necessarily leads to a small error on the test set. Adding hidden units will always lead to a reduction of the E_{learning}. However, adding hidden units will first lead to a reduction of the E_{test}, but then lead to an increase of E_{test}. This effect is called the peaking effect. The average learning and test error rates as a function of the learning set size are given in Fig. 12.10.

12.9 APPLICATIONS

Back-propagation has been applied to a wide variety of research applications.

- Sejnowski and Rosenberg (1986) produced a spectacular success with NETtalk, a system that converts printed English text into highly intelligible speech.
- A feed-forward network with one layer of hidden units has been described by Gorman and Sejnowski (1988) as a classification machine for sonar signals.

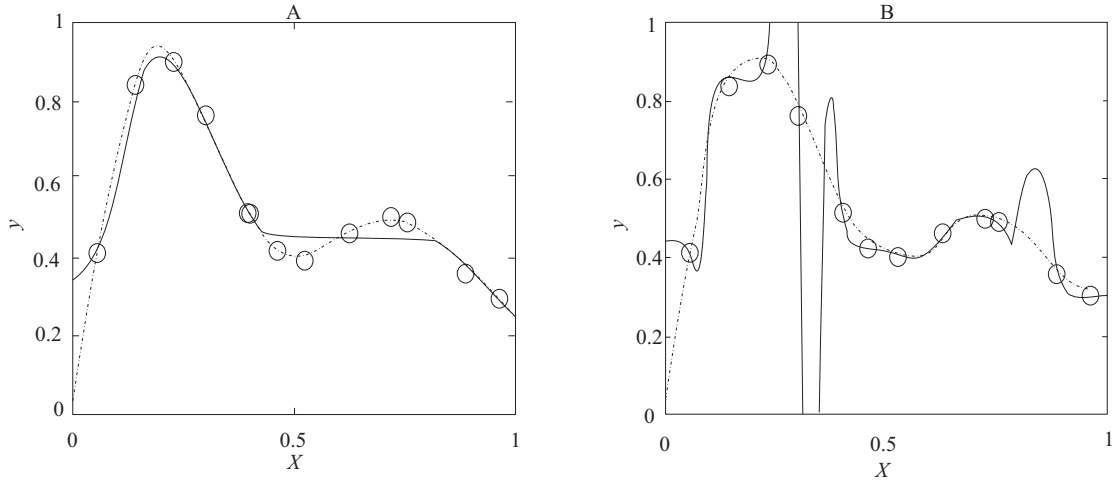

Fig. 12.9 Effect of the number of hidden units on the network performance. The dashed line gives the desired function, the circles denote the learning samples and the drawn line gives the approximation by the network. 12 learning samples are used. a) 5 hidden units. b) 20 hidden units.

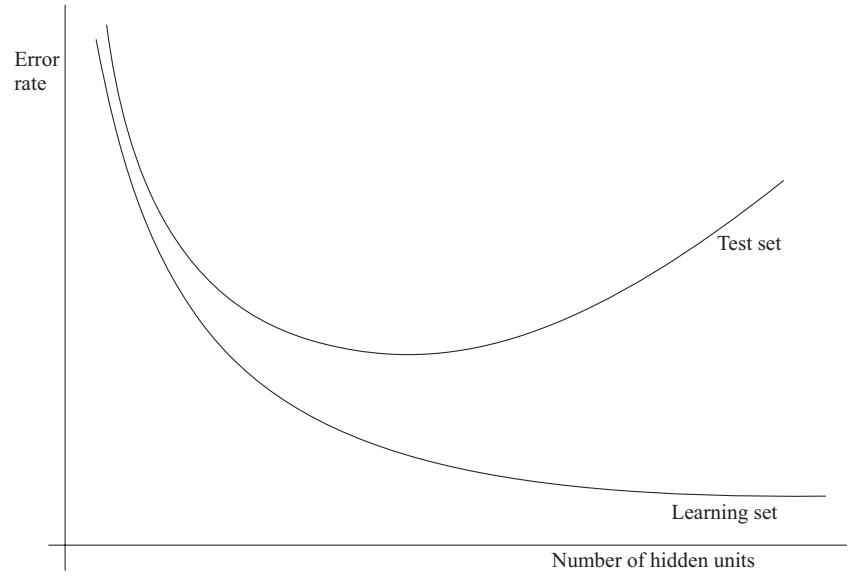

Fig. 12.10 The average learning error rate and the average test error rate as a function of the number of hidden units.

· A multi-layer feed-forward network with a back-propagation training algorithm is used to learn an unknown function between input and output signals from the presentation of examples. It is hoped that the network is able to generalize correctly, so that input values which are not presented as learning patterns will result in correct output values. An example is the work of Josin (1988), who used a two-layer feed-forward network with back-propagation learning to perform the inverse kinematic transform which is needed by a robot arm controller.

QUESTION BANK

1. Explain the multi-layer feed forward networks.
2. Describe the generalized delta rule.
3. What is back-propagation algorithm? Explain.
4. How the weights are adjusted with sigmoid activation function? Explain with an example.
5. Explain learning rate and momentum with back-propagation with an example.
6. Explain the sine activation function with an example.
7. What are the deficiencies of back-propagation algorithm?
8. Explain various methods employed to overcome the deficiencies of back-propagation algorithm.
9. How good are multi-layer feed forward networks? Explain.
10. Explain the effect of the number of learning samples in multi-layer feed forward networks.
11. Explain the effect of the number of hidden links in multi-layer feed forward networks.
12. What are the applications of back-propagation algorithm?

REFERENCES

1. M. Minsky, and S. Papert, *Perceptrons*: *An Introduction to Computational Geometry*, The MIT Press, 1969.
2. D.E. Rumelhart, G.E. Hinton, and R.J. Williams, Learning representations by back-propagating errors, *Nature*, Vol. 323, pp. 533-536, 1986.
3. D.B. Parker, Learning-Logic (Tech. Rep. Nos. TR (47), Cambridge, MA: Massachusetts Institute of Technology, Center for Computational Research in Economics and Management Science, 1985.
4. Y.L. Cun, Y. L, Une procedure d'apprentissage pour reseau a seuil assymetrique. *Proceedings of Cognitiva*, Vol. 85, pp. 599-604, 1985.
5. K. Hornik, M. Stinchcombe, and H. White, Multilayer feed forward networks are universal approximates, *Neural Networks*, Vol. 2, No. 5, pp. 359-366, 1989.
6. K.I. Funahashi, On the approximate realization of continuous mappings by neural networks, *Neural Networks*, Vol. 2, No. 3, Vol. 193-192, 1989.
7. G. Cybenko, Approximation by superpositions of a sigmoidal function. *Mathematics of Control*, *Signals*, *and Systems*, Vol. 2, No. 4, pp. 303-314, 1989.
8. E.J. Hartman, J.D. Keeler, and J.M. Kowalski, Layered neural networks with Gaussian hidden units as universal approximations, *Neural Computation*, Vol. 2, No. 2, pp. 210-215, 1990.
9. W.H. Press, B.P. Flannery, S.A. Teukolsky, and W.T. Vetterling, *Numerical Recipes*: *The Art of Scientific Computing*, Cambridge: Cambridge University Press, 1986.
10. J. Stoer, and R. Bulirsch, *Introduction to Numerical Analysis*, New York-Heidelberg- Berlin: Springer-Verlag, 1980.

11. M.R. Hestenes, and E. Stiefel, Methods of conjugate gradients for solving linear systems, *Journal of National Bureau of Standards*, Vol. 49, pp. 409-436, 1952.
12. E. Polak, *Computational Methods in Optimization*, New York: Academic Press, 1971.
13. M.J.D. Powell, Restart procedures for the conjugate gradient method, *Mathematical Programming*, Vol. 12, pp. 241-254, 1977.
14. T.J. Sejnowski, and C.R. Rosenberg, *NETtalk*: *A Parallel Network that Learns to Read Aloud* (Tech. Rep. Nos. JHU/EECS-86/01), The John Hopkins University Electrical Engineering and Computer Science Department, 1986.
15. R.P. Gorman, and T.J. Sejnowski, Analysis of hidden units in a layered network trained to classify sonar targets, *Neural Networks*, Vol. 1, No. 1, pp. 75-89, 1988.
16. G. Josin, Neural-space generalization of a topological transformation, *Biological Cybernetics*, Vol. 59, pp. 283-290, 1988.

CHAPTER 13

Recurrent Networks

13. 1 INTRODUCTION

The learning algorithms discussed in the previous chapter were applied to feed-forward networks: all data flows in a network in which no cycles are present.

But what happens when we introduce a cycle? For instance, we can connect a hidden unit with itself over a weighted connection, connect hidden units to input units, or even connect all units with each other. Although, as we know from the previous chapter, the approximation capabilities of such networks do not increase, we may obtain decreased complexity, network size, etc. to solve the same problem.

An important question we have to consider is the following: what do we want to learn in a recurrent network? After all, when one is considering a recurrent network, it is possible to continue propagating activation values until a stable point (attractor) is reached. As we will see in the sequel, there exist recurrent network, which are attractor based, i.e., the activation values in the network are repeatedly updated until a stable point is reached after which the weights are adapted, but there are also recurrent networks where the learning rule is used after each propagation (where an activation value is transversed over each weight only once), while external inputs are included in each propagation. In such networks, the recurrent connections can be regarded as extra inputs to the network (the values of which are computed by the network itself).

In this chapter, recurrent extensions to the feed-forward network will be discussed. The theory of the dynamics of recurrent networks extends beyond the scope of a one-semester course on neural networks. Yet the basics of these networks will be discussed.

Also some special recurrent networks will be discussed: the Hopfield network, which can be used for the representation of binary patterns; subsequently we touch upon Boltzmann machines, therewith introducing stochasticity in neural computation.

13.2 THE GENERALISED DELTA - RULE IN RECURRENT NETWORKS

The back-propagation learning rule, introduced in chapter 12, can be easily used for training patterns in recurrent networks. Before we will consider this general case, however, we will first describe networks

where some of the hidden unit activation values are fed back to an extra set of input units (the Elman network), or where output values are fed back into hidden units (the Jordan network).

A typical application of such a network is the following. Suppose we have to construct a network that must generate a control command depending on an external input, which is a time series $x(t)$, $x(t-1)$, $x(t-2)$, …. With a feed-forward network there are two possible approaches:

1. Create inputs x_1, x_2, …, x_n which constitute the last n values of the input vector. Thus a 'time window' of the input vector is input to the network.
2. Create inputs x, x', x'', …. Besides only inputting $x(t)$, we also input its first, second, etc. derivatives. Naturally, computation of these derivatives is not a trivial task for higher-order derivatives.

The disadvantage is, of course, that the input dimensionality of the feed-forward network is multiplied with n, leading to a very large network, which is slow and difficult to train. The Jordan and Elman networks provide a solution to this problem. Due to the recurrent connections, a window of inputs need not be input anymore; instead, the network is supposed to learn the influence of the previous time steps itself.

13.2.1 The Jordan Network

One of the earliest recurrent neural networks was the Jordan network. An example of this network is shown in Fig. 13.1. In the Jordan network, the activation values of the output units are fed back into the

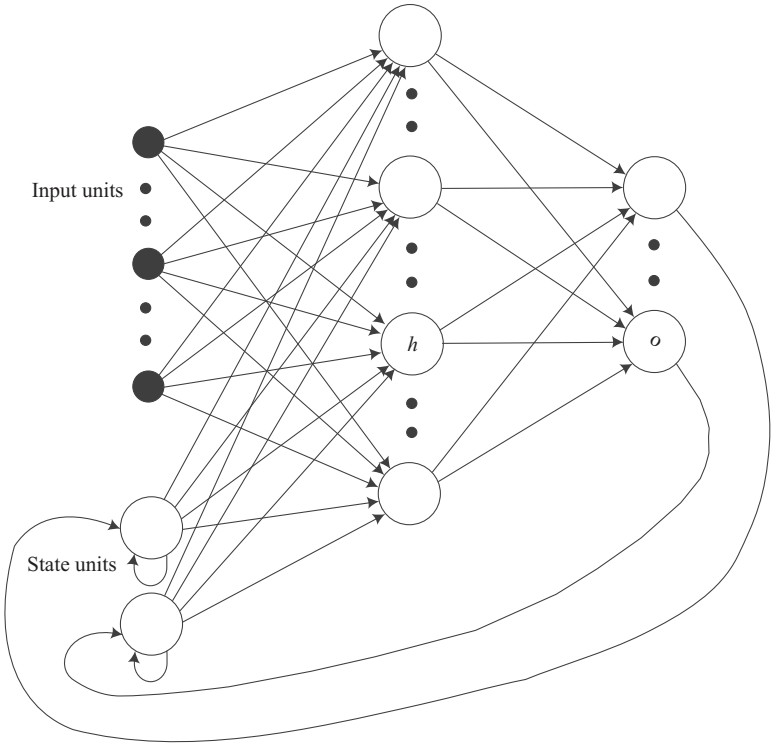

Fig. 13.1 The Jordan network. Output activation values are fed back to the input layer, to a set of extra neurons called the state units.

input layer through a set of extra input units called the state units. There are as many state units as there are output units in the network. The connections between the output and state units have a fixed weight of +1; learning takes place only in the connections between input and hidden units as well as hidden and output units. Thus all the learning rules derived for the multi-layer perceptron can be used to train this network.

13.2.2 The Elman Network

In the Elman network a set of context units are introduced, which are extra input units whose activation values are fed back from the hidden units. Thus the network is very similar to the Jordan network, except that (1) the hidden units instead of the output units are fed back; and (2) the extra input units have no self-connections.

The schematic structure of this network is shown in Fig. 13.2.

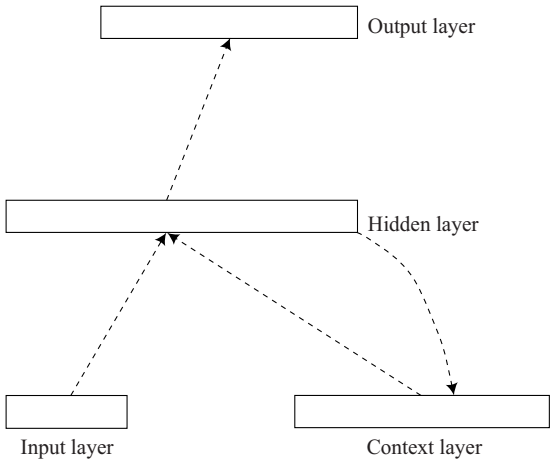

Fig. 13.2 The Elman network. With this network, the hidden unit activation values are fed back to the input layer, to a set of extra neurons called the context units.

Again the hidden units are connected to the context units with a fixed weight of value +1. Learning is done as follows:

1. The context units are set to 0; $t = 1$
2. Pattern x^t is clamped, the forward calculations are performed once;
3. The back-propagation learning rule is applied;
4. $t \leftarrow t + 1$; go to 2.

The context units at step t thus always have the activation value of the hidden units at step $t - 1$.

Example 13.1: As we mentioned above, the Jordan and Elman networks can be used to train a network on reproducing time sequences. The idea of the recurrent connections is that the network is able to 'remember' the previous states of the input values. As an example, we trained an Elman network on controlling an object moving in 1 D. This object has to follow a pre-specified trajectory x_d. To control the object, forces F must be applied, since the object suffers from friction and perhaps other external forces.

To tackle this problem, we use an Elman net with inputs x and x_d, one output F, and three hidden units. The hidden units are connected to three context units. In total, five units feed into the hidden layer.

The results of training are shown in Fig. 13.3. The same test can be done with an ordinary feed-forward network with sliding window input. We tested this with a network with five inputs, four of

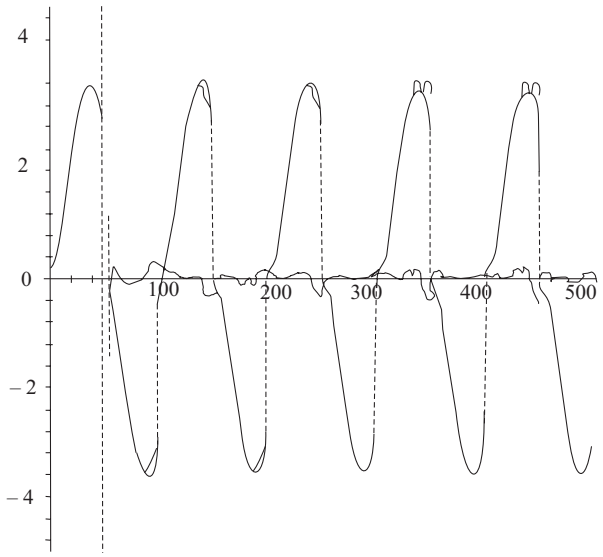

Fig. 13.3 Training an Elman network to control an object. The solid line depicts the desired trajectory x_d; the dashed line the realized trajectory. The third line is the error.

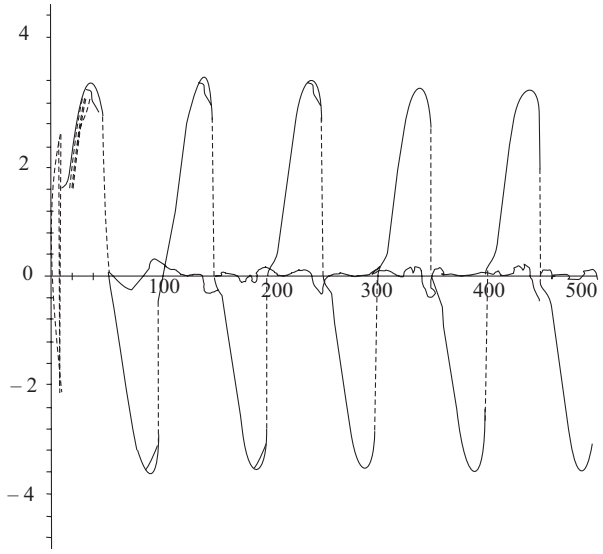

Fig. 13.4 Training a feed-forward network to control an object. The solid line depicts the desired trajectory x_d; the dashed line the realized trajectory. The third line is the error.

which constituted the sliding window x_{-3}, x_{-2}, x_{-1} and x_0, and one the desired next position of the object. Results are shown in Fig. 13.4. The disappointing observation is that the results are actually better with the ordinary feed-forward network, which has the same complexity as the Elman network.

13.2.3 Back-Propagation in Fully Recurrent Networks

More complex schemes than the above are possible. For instance, independently of each other Pineda (1987) and Almeida (1987) discovered that error back-propagation is in fact a special case of a more general gradient learning method, which can be used for training attractor networks. However, also when a network does not reach a fixed point, a learning method can be used: back-propagation through time (Pearlmutter, 1989, 1990). This learning method, the discussion of which extents beyond the scope of our course, can be used to train a multi-layer perceptron to follow trajectories in its activation values.

13.3 THE HOPFIELD NETWORK

One of the earliest recurrent neural networks reported in literature was the auto-associator independently described by Anderson (1977) and Kohonen (1977). It consists of a pool of neurons with connections between each unit i and j, $i \neq j$ (see Fig. 15.5). All connections are weighted.

Hopfiled (1982) brings together several earlier ideas concerning these networks and presents a complete mathematical analysis.

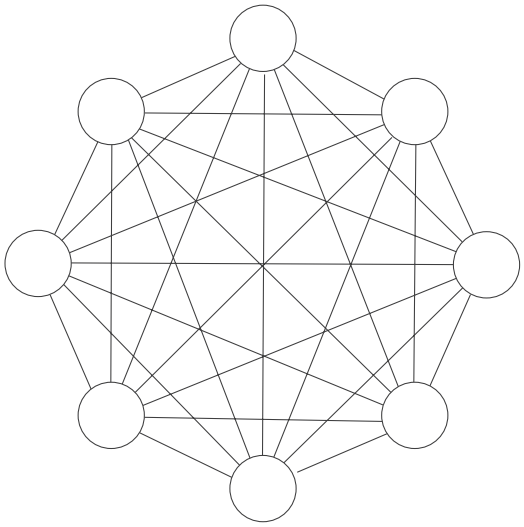

Fig. 13.5 The auto-associator network. All neurons are both input and output neurons, i.e., a pattern is clamped, the network iterates to a stable state, and the output of the network consists of the new activation values of the neurons.

13.3.1 Description

The Hopfield network consists of a set of N interconnected neurons (Fig. 13.5), which update their activation values asynchronously and independently of other neurons. All neurons are both input and output neurons. The activation values are binary. Originally, Hopfield chose activation values of 1 and 0, but using values +1 and –1 presents some advantages discussed below. We will therefore adhere to the latter convention.

The state of the system is given by the activation values $Y = y(k)$. The net input $S_k(t + 1)$ of a neuron k at cycle $t + 1$ is a weighted sum

$$S_k(t+1) = \sum_{j \neq k} y_j(t) w_{jk} + \theta_k \qquad ...(13.1)$$

A simple threshold function (Fig. 10.2) is applied to the net input to obtain the new activation value $y_i(t + 1)$ at time $t + 1$:

$$y_k(t+1) = \begin{cases} +1 & \text{if } S_k(t+1) > U_k \\ -1 & \text{if } S_k(t+1) < U_k \\ y_k(t) & \text{otherwise} \end{cases} \qquad ...(13.2)$$

i.e., $yk(t + 1) = \text{sgn}(S_k(t + 1))$ For simplicity we henceforth choose $U_k = 0$, but this is of course not essential.

A neuron k in the Hopfield network is called stable at time t if, in accordance with equations (13.1) and (13.2),

$$y_k(t) = \text{sgn}(S_k(t-1)) \qquad ...(13.3)$$

A state α is called stable if, when the network is in state α, all neurons are stable. A pattern x^p is called stable if, when x^p is clamped, all neurons are stable.

When the extra restriction $w_{jk} = w_{kj}$ is made, the behavior of the system can be described with an energy function

$$\varepsilon = -\frac{1}{2} \sum_{j \neq k} \sum \sum y_j \, y_k \, w_{jk} - \sum_k \theta_k y_k \qquad ...(13.4)$$

Theorem 13.1: A recurrent network with connections $w_{jk} = w_{kj}$ in which the neurons are updated using rule (13.2) has stable limit points.

Proof: First, note that the energy expressed in eq. (13.4) is bounded from below, since the y_k are bounded from below and the w_{jk} and θ_k are constant. Secondly, ε is monotonically decreasing when state changes occur, because

$$\Delta\varepsilon = -\Delta y_k \left(\sum_{j \neq k} y_j w_{jk} + \theta_k \right) \qquad ...(13.5)$$

is always negative when y_k changes according to eqs. (13.1) and (13.2).

The advantage of a + 1/–1 model over a 1/0 model then is symmetry of the states of the network. For, when some pattern x is stable, its inverse is stable, too, whereas in the 1/0 model this is not always true (as an example, the pattern 00 … 00 is always stable, but 11 … 11 need not be). Similarly, both a pattern and its inverse have the same energy in the +1/–1 model.

Removing the restriction of bidirectional connections (i.e., $w_{jk} = w_{kj}$) results in a system that is not guaranteed to settle to a stable state.

13.3.2 Hopfield Network as Associative Memory

A primary application of the Hopfield network is an associative memory. In this case, the weights of the connections between the neurons have to be thus set that the states of the system corresponding with the patterns which are to be stored in the network are stable. These states can be seen as 'dips' in energy space. When the network is cued with a noisy or incomplete test pattern, it will render the incorrect or missing data by iterating to a stable state, which is in some sense 'near' to the cued pattern.

The Hebb rule can be used to store P patterns:

$$w_{jk} = \begin{cases} \sum_{p=1}^{P} x_j^p x_k^p & \text{if } j \neq k \\ 0 & \text{otherwise} \end{cases} \qquad ...(13.6)$$

i.e., if x_j^p and x_k^p are equal, w_{jk} is increased, otherwise decreased by one (note that, in the original Hebb rule, weights only increase). It appears, however, that the network gets saturated very quickly, and that about $0{:}15N$ memories can be stored before recall errors become severe.

There are two problems associated with storing too many patterns:

1. The stored patterns become unstable;
2. Spurious stable states appear (i.e., stable states which do not correspond with stored patterns).

The first of these two problems can be solved by an algorithm proposed by Bruce et al. (Bruce, Canning, Forrest, Gardner, & Wallace, 1986).

Algorithm 13.1: Given a starting weight matrix $W = [w_{jk}]$, for each pattern x^p to be stored and each element x_k^p in x^p define a correction ε_k such that

$$\varepsilon_k = \begin{cases} 0 & \text{if } y_k \text{ is stable and } x^p \text{ is clamped} \\ 1 & \text{otherwise} \end{cases} \qquad ...(13.7)$$

Now modify w_{jk} by $\Delta w_{jk} = y_j\, y_k(\varepsilon_j + \varepsilon_k)$ if $j \neq k$. Repeat this procedure until all patterns are stable.

It appears that, in practice, this algorithm usually converges. There exist cases, however, where the algorithm remains oscillatory (try to find one)!

The second problem stated above can be alleviated by applying the Hebb rule in reverse to the spurious stable state, but with a low learning factor (Hopfield, Feinstein, & Palmer, 1983). Thus these patterns are weakly unstored and will become unstable again.

13.3.3 Neurons with Graded Response

The network described in section 13.3.1 can be generalized by allowing continuous activation values. Here, the threshold activation function is replaced by a sigmoid. As before, this system can be proved to be stable when a symmetric weight matrix is used (Hopfield, 1984).

13.3.4 Hopfield Networks for Optimization Problems

An interesting application of the Hopfield network with graded response arises in a heuristic solution to the NP-complete traveling salesman problem (Garey & Johnson, 1979). In this problem, a path of minimal distance must be found between n cities, such that the begin- and end-points are the same.

Hopfield and Tank (1985) use a network with $n \times n$ neurons. Each row in the matrix represents a city, whereas each column represents the position in the tour. When the network is settled, each row and each column should have one and only one active neuron, indicating a specific city occupying a specific position in the tour. The neurons are updated using rule (13.2) with a sigmoid activation function between 0 and 1. The activation value $y_{xj} = 1$ indicates that city X occupies the j^{th} place in the tour.

An energy function describing this problem can be set up as follows. To ensure a correct solution, the following energy must be minimized:

$$\varepsilon = \frac{A}{2}\sum_X\sum_j\sum_{k\neq j} y_{Xj}y_{Xk} + \frac{B}{2}\sum_j\sum_X\sum_{X\neq Y} y_{Xj}y_{Yj} + \frac{C}{2}\left(\sum_X\sum_j y_{Xj} - n\right)^2 \qquad ...(13.8)$$

where A, B, and C are constants. The first and second terms in equation (13.8) are zero if and only if there is a maximum of one active neuron in each row and column, respectively. The last term is zero if and only if there are exactly n active neurons.

To minimise the distance of the tour, an extra term

$$\varepsilon = \frac{D}{2}\sum_X\sum_{Y\neq X}\sum_j d_{XY}y_{Xj}\left(y_{Y,\,j+1} + y_{Y,\,j-1}\right) \qquad ...(13.9)$$

is added to the energy, where d_{XY} is the distance between cities X and Y and D is a constant. For convenience, the subscripts are defined modulo n.

The weights are set as follows:

$$\begin{aligned} w_{Xj,\,Yk} &= -A\delta_{XY}(1-\delta_{jk}) \text{ inhibitory connections within each row} \\ &= -B\delta_{jk}(1-\delta_{XY}) \text{ inhibitory connections within each column} \\ &= -C \text{ global inhibition} \\ &= -Dd_{XY}(\delta_{k,\,j+1} + \delta_{k,\,j-1}) \text{ data term} \end{aligned} \qquad ...(13.10)$$

where $\delta_{jk} = 1$ if $j = k$ and 0 otherwise. Finally, each neuron has an external bias input C_n.

Although this application is interesting from a theoretical point of view, the applicability is limited. Whereas Hopfield and Tank state that the network converges to a valid solution in 16 out of 20 trials while 50% of the solutions are optimal, other reports show less encouraging results. For example, (Wilson and Pawley, 1988) find that in only 15% of the runs a valid result is obtained, few of which lead

to an optimal or near-optimal solution. The main problem is the lack of global information. Since, for an N-city problem, there are $N!$ possible tours, each of which may be traversed in two directions as well as started in N points, the number of different tours is $N!/2N$. Differently put, the N-dimensional hypercube in which the solutions are situated is $2N$ degenerate. The degenerate solutions occur evenly within the hypercube, such that all but one of the final $2N$ configurations are redundant. The competition between the degenerate tours often leads to solutions which are piecewise optimal but globally inefficient.

13.4 BOLTZMANN MACHINES

The Boltzmann machine, as first described by Ackley, Hinton, and Sejnowski in 1985 is a neural network that can be seen as an extension to Hopfield networks to include hidden units, and with a stochastic instead of deterministic update rule. The weights are still symmetric. The operation of the network is based on the physics principle of annealing. This is a process whereby a material is heated and then cooled very, very slowly to a freezing point. As a result, the crystal lattice will be highly ordered, without any impurities, such that the system is in a state of very low energy. In the Boltzmann machine this system is mimicked by changing the deterministic update of equation (13.2) in a stochastic update, in which a neuron becomes active with a probability p,

$$p(y_k \leftarrow +1) = \frac{1}{1 + e^{-\Delta\varepsilon_k / T}} \qquad ...(13.11)$$

where T is a parameter comparable with the (synthetic) temperature of the system. This stochastic activation function is not to be confused with neurons having a sigmoid deterministic activation function.

In accordance with a physical system obeying a Boltzmann distribution, the network will eventually reach 'thermal equilibrium' and the relative probability of two global states α and β will follow the Boltzmann distribution

$$\frac{P_\alpha}{P_\beta} = e^{-(\varepsilon_\alpha - \varepsilon_\beta)/T} \qquad ...(13.12)$$

where P_α is the probability of being in the α^{th} global state, and ε_α is the energy of that state. Note that at thermal equilibrium the units still change state, but the probability of finding the network in any global state remains constant.

At low temperatures there is a strong bias in favor of states with low energy, but the time required to reach equilibrium may be long. At higher temperatures the bias is not so favorable but equilibrium is reached faster. A good way to beat this trade-off is to start at a high temperature and gradually reduce it. At high temperatures, the network will ignore small energy differences and will rapidly approach equilibrium. In doing so, it will perform a search of the coarse overall structure of the space of global states, and will find a good minimum at that coarse level. As the temperature is lowered, it will begin to respond to smaller energy differences and will find one of the better minima within the coarse-scale minimum it discovered at high temperature.

As multi-layer perceptions, the Boltzmann machine consists of a non-empty set of visible and a possibly empty set of hidden units. Here, however, the units are binary-valued and are updated stochastically and asynchronously. The simplicity of the Boltzmann distribution leads to a simple learning procedure, which adjusts the weights so as to use the hidden units in an optimal way (Ackley et al., 1985). This algorithm works as follows:

- First, the input and output vectors are clamped.
- The network is then annealed until it approaches thermal equilibrium at a temperature of 0. It then runs for a fixed time at equilibrium and each connection measures the fraction of the time during which both the units it connects are active. This is repeated for all input-output pairs so that each connection can measure $(y_j\, y_k)^{\text{clamped}}$, the expected probability, averaged over all cases, that units j and k are simultaneously active at thermal equilibrium when the input and output vectors are clamped.
- Similarly, $(y_j\, y_k)^{\text{free}}$ is measured when the output units are not clamped but determined by the network.
- In order to determine optimal weights in the network, an error function must be determined. Now, the probability $P^{\text{free}}(Y^p)$ that the visible units are in state $Y^p$when the system is running freely can be measured. Also, the desired probability $P^{\text{clamped}}(Y^p)$that the visible units are in state (Y^p) is determined by clamping the visible units and letting the network run.
- Now, if the weights in the network are correctly set, both probabilities are equal to each other, and the error E in the network must be 0. Otherwise, the error must have a positive value measuring the discrepancy between the network's internal mode and the environment. For this effect, the 'asymmetric divergence' or 'Kullback information' is used:

$$E = \sum_p P^{\text{clamped}}(Y^p) \log \frac{P^{\text{clamped}}(Y^p)}{P^{\text{free}}(Y^P)} \qquad ...(13.13)$$

Now, in order to minimize E using gradient descent, we must change the weights according to

$$\Delta w_{jk} = -\gamma \frac{\partial E}{\partial w_{jk}} \qquad ...(13.14)$$

It is not difficult to show that

$$\frac{\partial E}{\partial w_{jk}} = -\frac{1}{T}\left[(y_j y_k)^{\text{clamped}} - (y_j y_k)^{\text{free}}\right] \qquad ...(13.15)$$

Therefore, each weight is updated by

$$\Delta w_{jk} = \gamma\left[(y_j y_k)^{\text{clamped}} - (y_j y_k)^{\text{free}}\right] \qquad ...(13.16)$$

QUESTION BANK

1. What happens when a cyclic data is introduced to feed forward networks?
2. Explain the generalized delta-rule in recurrent networks.
3. Describe the Jordan network with an example.
4. Describe Elman network with an example.
5. Describe the Hopfield network.
6. Describe the Hopfield network as associative memory.
7. Describe Hopfield network for optimization problems.
8. Describe the Boltzman machine.
9. What are the problems resulted while storing too many patterns using associative memory? How these problems can be solved?

REFERENCES

1. M.I. Jordan, Attractor dynamics and parallelism in a connectionist sequential machine, In Proceedings of the Eighth Annual Conference of the Cognitive Science Society, Hillsdale, NJ: Erlbaum, pp. 531-546, 1986.
2. M.I. Jordan, Serial Order: A Parallel Distributed Processing Approach (Tech. Rep. No. 8604). San Diego, La Jolla, CA: Institute for Cognitive Science, University of California, 1986.
3. J.L. Elman, Finding structure in time. *Cognitive Science*, Vol. 14, pp. 179-211, 1990.
4. F. Pineda, Generalization of back-propagation to recurrent neural networks, *Physical Review Letters*, Vol. 19, and pp. 2229-2232, 1987.
5. L.B. Almeida, A learning rule for asynchronous perceptrons with feedback in a combinatorial environment, In *Proceedings of the First International Conference on Neural Networks*, Vol. 2, pp. 609-618,1987.
6. B.A. Pearlmutter, Learning state space trajectories in recurrent neural networks, *Neural Computation*, Vol. 1, No. 2, pp. 263-269, 1989.
7. B.A. Pearlmutter, *Dynamic Recurrent Neural Networks* (Tech. Rep. Nos. CMU-CS-90-196), Pittsburgh, PA 15213: School of Computer Science, Carnegie Mellon University, 1990.
8. J.A. Anderson, *Neural Models with Cognitive Implications*. In D. LaBerge and S.J. Samuels (Eds.), Basic Processes in Reading Perception and Comprehension Models, Hillsdale, NJ: Erlbaum, pp. 27-90, 1977.
9. T. Kohonen, *Associative Memory*: *A System-Theoretical Approach*, Springer-Verlag, 1977.
10. J.J. Hopfield, Neural networks and physical systems with emergent collective computational abilities, *Proceedings of the National Academy of Sciences*, Vol. 79, pp. 2554-2558, 1982.
11. A.D. Bruce, A. Canning, B. Forrest, E. Gardner, and D.J. Wallace, Learning and memory properties in fully connected networks, In J.S. Denker (Ed.), *AIP Conference Proceedings 151*, *Neural Networks for Computing*, pp. 65-70, DUNNO, 1986.

12. J.J. Hopfield, D.I. Feinstein, and R.G. Palmer, unlearning has a stabilizing effect in collective memories, *Nature*, Vol. 304, pp. 159-159, 1983.
13. J.J. Hopfield, Neurons with graded response have collective computational properties like those of two-state neurons, *Proceedings of the National Academy of Sciences*, Vol. 81, pp. 3088-3092, 1984.
14. M.R. Garey, and D.S. Johnson, *Computers and Intractability*. New York: W.H. Freeman, 1979.
15. J.J. Hopfield, and D.W. Tank, neural computation of decisions in optimization problems, *Biological Cybernetics*, Vol. 52, pp. 141-152, 1985.
16. G.V. Wilson, and G.S. Pawley, On the stability of the traveling salesman problem algorithm of Hopfield and tank, *Biological Cybernetics*, Vol. 58, pp. 63-70, 1988.
17. D.H. Ackley, G.E. Hinton, and T.J. Sejnowski, (1985). A learning algorithm for Boltzmann machines, *Cognitive Science*, Vol. 9, No. 1, pp. 147-169, 1985.

CHAPTER 14

Self-Organizing Networks

14. 1 INTRODUCTION

In the previous chapters we discussed a number of networks, which were trained to perform a mapping F: $\Re'' \rightarrow \Re^m$ by presenting the network 'examples' (x^p, d^p) with $d^p = F(x^p)$ of this mapping. However, problems exist where such training data, consisting of input and desired output pairs are not available, but where the only information is provided by a set of input patterns x^p. In these cases the relevant information has to be found within the (redundant) training samples x^p.

Some examples of such problems are:

- **Clustering:** the input data may be grouped in 'clusters' and the data processing system has to find these inherent clusters in the input data. The output of the system should give the cluster label of the input pattern (discrete output);
- **Vector quantisation:** this problem occurs when a continuous space has to be discretized. The input of the system is the n-dimensional vector x, the output is a discrete representation of the input space. The system has to find optimal discretization of the input space;
- **Dimensionality reduction:** the input data are grouped in a subspace, which has lower dimensionality than the dimensionality of the data. The system has to learn an optimal mapping, such that most of the variance in the input data is preserved in the output data;
- **Feature extraction:** the system has to extract features from the input signal. This often means a dimensionality reduction as described above.

In this chapter we discuss a number of neuro-computational approaches for these kinds of problems. Training is done without the presence of an external teacher. The unsupervised weight adapting algorithms are usually based on some form of global competition between the neurons.

There are very many types of self-organizing networks, applicable to a wide area of problems. One of the most basic schemes is competitive learning as proposed by Rumelhart and Zipser (1985). A very similar network but with different emergent properties is the topology-conserving map devised by Kohonen. Other self-organizing networks are ART, proposed by Carpenter and Grossberg (1987), and Fukushima (1975).

14.2 COMPETITIVE LEARNING

14.2.1 Clustering

Competitive learning is a learning procedure that divides a set of input patterns in clusters that are inherent to the input data. A competitive learning network is provided only with input vectors x and thus implements an unsupervised learning procedure. We will show its equivalence to a class of 'traditional' clustering algorithms shortly. Another important use of these networks is vector quantisation.

An example of a competitive learning network is shown in Fig. 14.1. All output units o are connected to all input units i with weights w_{io}. When an input pattern x is presented, only a single output unit of the network (the winner) will be activated. In a correctly trained network, all x in one cluster will have the same winner. For the determination of the winner and the corresponding learning rule, two methods exist.

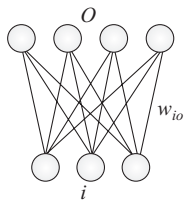

Fig. 14.1 A simple competitive learning network. Each of the four outputs *o* is connected to all inputs *i*.

Winner Selection: Dot Product

For the time being, we assume that both input vectors x and weight vectors w_o are normalized to unit length. Each output unit o calculates its activation value y_o according to the dot product of input and weight vector:

$$y_o = \sum_i w_{io} x_i = w_o^T x \qquad ...(14.1)$$

In a next pass, output neuron k is selected with maximum activation

$$\forall_o \neq k : y_o \leq y_k \qquad ...(14.2)$$

Activations are reset such that $y_k = 1$ and $y_{o \neq k} = 0$. This is the competitive aspect of the network, and we refer to the output layer as the winner-take-all layer. The winner-take-all layer is usually implemented in software by simply selecting the output neuron with highest activation value. This function can also be performed by a neural network known as MAXNET (Lippmann, 1989). In MAXNET, all neurons o are connected to other units $o0$ with inhibitory links and to itself with an excitatory link:

$$w_{o,o} = \begin{cases} -\varepsilon & \text{if } o \neq o' \\ +1 & \text{otherwise} \end{cases} \qquad ...(14.3)$$

It can be shown that this network converges to a situation where only the neuron with highest initial activation survives, whereas the activations of all other neurons converge to zero. From now on, we will simply assume a winner k is selected without being concerned which algorithm is used.

Once the winner k has been selected, the weights are updated according to:

$$w_k(t+1) = \frac{w_k(t) + \gamma(x(t) - w_k(t))}{\|w_k(t) + \gamma(x(t) - w_k(t))\|} \quad ...(14.4)$$

where the divisor ensures that all weight vectors w are normalized. Note that only the weights of winner k are updated.

The weight update given in equation (14.4) effectively rotates the weight vector w_o towards the input vector x. Each time an input x is presented; the weight vector closest to this input is selected and is subsequently rotated towards the input. Consequently, weight vectors are rotated towards those areas where many inputs appear: the clusters in the input. This procedure is visualized in Fig. 14.2.

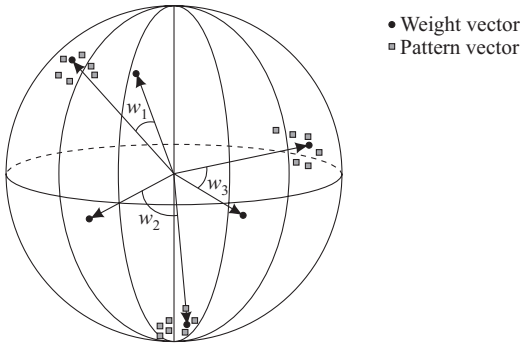

Fig. 14.2 Example of clustering in 3*D* with normalized vectors, which all lie on the unity sphere. The three weight vectors are rotated towards the centers of gravity of the three different input clusters.

Winner selection: Euclidean distance

Previously it was assumed that both inputs x and weight vectors w were normalized. Using the activation function given in equation (14.1) gives a 'biological plausible' solution. In Fig. 14.3 it is shown how the algorithm would fail if normalized vectors were to be used. Naturally one would like to accommodate the algorithm for normalized input data. To this end, the winning neuron k is selected with its weight vector w_k closest to the input pattern x, using the Euclidean distance measure:

$$k: \|w_k - x\| \leq \|w_o - x\| \ \forall_o \quad ...(14.5)$$

It is easily checked that equation (14.5) reduces to (14.1) and (14.2) if all vectors are normalized. The Euclidean distance norm is therefore a more general case of equations (14.1) and (14.2). Instead of rotating the weight vector towards the input as performed by equation (14.4), the weight update must be changed to implement a shift towards the input:

$$w_k(t+1) = w_k(t) + \gamma(x(t) - w_k(t)) \quad ...(14.6)$$

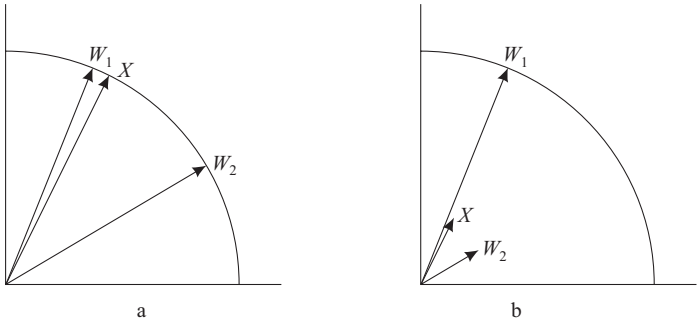

Fig. 14.3 Determining the winner in a competitive learning network. a. Three normalized vectors. b. The three vectors having the same directions as in a., but with different lengths. In *a*., vectors x and w_1 are nearest to each other, and their dot product $x^T w_1 = |x||w_1| \cos \alpha$ is larger than the dot product of x and w_2. In b., however, the pattern and weight vectors are not normalized, and in this case w_2 should be considered the 'winner' when x is applied. However, the dot product $x^T w_1$ is still larger than $x^T w_2$.

Again only the weights of the winner are updated.

A point of attention in these recursive clustering techniques is the initialization. Especially if the input vectors are drawn from a large or high-dimensional input space, it is not beyond imagination that a randomly initialized weight vector w_o will never be chosen as the winner and will thus never be moved and never be used. Therefore, it is customary to initialize weight vectors to a set of input patterns $\{x\}$ drawn from the input set at random. Another more thorough approach that avoids these and other problems in competitive learning is called leaky learning. This is implemented by expanding the weight update given in equation (14.6) with

$$w_l(t+1) = w_l(t) + \gamma'(x(t) - w_l(t)) \; \forall_l \neq k \qquad ...(14.7)$$

with $\gamma' < \gamma$ the leaky learning rate. A somewhat similar method is known as frequency sensitive competitive learning (Ahalt, Krishnamurthy, Chen, & Melton, 1990). In this algorithm, each neuron records the number of times it is selected winner. The more often it wins, the less sensitive it becomes to competition. Conversely, neurons that consistently fail to win increase their chances of being selected winner.

Cost function: Earlier it was claimed, that a competitive network performs a clustering process on the input data. i.e., input patterns are divided in disjoint clusters such that similarities between input patterns in the same cluster are much bigger than similarities between inputs in different clusters. Similarity is measured by a distance function on the input vectors, as discussed before. A common criterion to measure the quality of a given clustering is the square error criterion, given by

$$E = \sum_p \|w_k - x^p\|^2 \qquad ...(14.8)$$

where k is the winning neuron when input x_p is presented. The weights w are interpreted as cluster centres. It is not difficult to show that competitive learning indeed seeks to find a minimum for this square error by the negative gradient of the error-function.

Theorem 14.1: The error function for pattern x^p

$$E^p = \sum_p \|w_k - x^p\|^2 \qquad ...(14.9)$$

where k is the winning unit, is minimised by the weight update rule in eq. (14.6).

Proof: As in eq. (3.12), we calculate the effect of a weight change on the error function. So we have that

$$\Delta_p W_{io} = -\gamma \frac{\partial E^p}{\partial w_{io}} \qquad ...(14.10)$$

where γ is a constant of proportionality. Now, we have to determine the partial derivative of E^p:

$$\frac{\partial E^p}{\partial w_{io}} = \begin{cases} w_{io} - x_i^p & \text{if unit } o \text{ wins} \\ 0 & \text{otherwise} \end{cases} \qquad ...(14.11)$$

such that

$$\Delta_p w_{io} = -\gamma(w_{io} - x_i^p) = \gamma(x_o^p - w_{io}) \qquad ...(14.12)$$

which is eq. (14.6) written down for one element of w_o.

Therefore, eq. (14.8) is minimized by repeated weight updates using eq. (14.6).

Example 14.1: In Fig. 14.4, 8 clusters of each 6 data points are depicted. A competitive learning network using Euclidean distance to select the winner was initialized with all weight vectors $w_o = 0$. The network was trained with $\gamma = 0{:}1$ and a $\gamma' = 0{:}001$ and the positions of the weights after 500 iterations are shown.

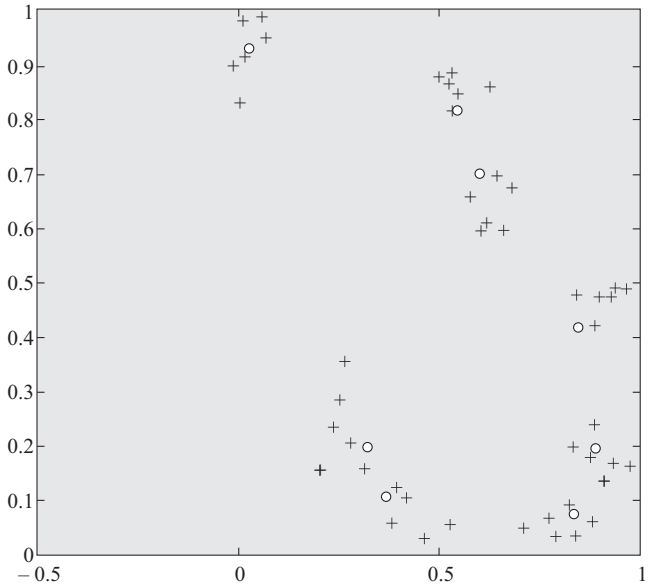

Fig. 14.4 Competitive learning for clustering data. The data are given by "+". The positions of the weight vectors after 500 iterations is given by "*o*".

14.2.2 Vector Quantisation

Another important use of competitive learning networks is found in vector quantisation. A vector quantisation scheme divides the input space in a number of disjoint subspaces and represents each input vector x by the label of the subspace it falls into (i.e., index k of the winning neuron). The difference with clustering is that we are not so much interested in finding clusters of similar data, but more in quantising the entire input space. The quantisation performed by the competitive learning network is said to 'track the input probability density function': the density of neurons and thus subspaces is highest in those areas where inputs are most likely to appear, whereas a more coarse quantisation is obtained in those areas where inputs are scarce. An example of tracking the input density is sketched in Figure 14.5. Vector quantisation through competitive learning results in a more fine-grained discretization in those areas of the input space where most input have occurred in the past.

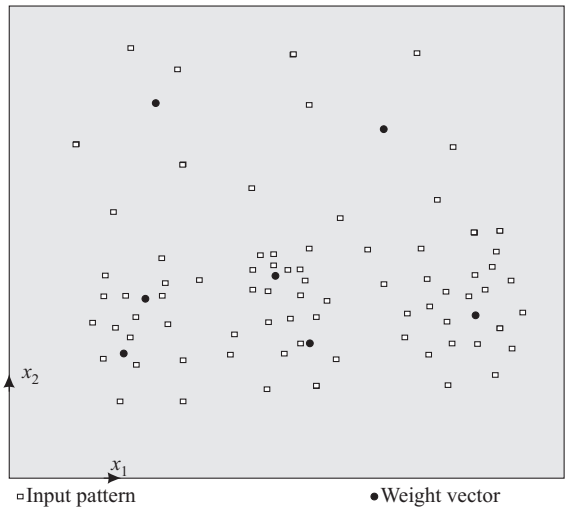

Fig. 14.5 This figure visualizes the tracking of the input density. The input patterns are drawn from $\Re^2$; the weight vectors also lie in $\Re^2$. In the areas where inputs are scarce, the upper part of the figure, only few (in this case two) neurons are used to discretized the input space. Thus, the upper part of the input space is divided into two large separate regions. The lower part, however, where many more inputs have occurred, five neurons discretized the input space into five smaller subspaces.

In this way, competitive learning can be used in applications where data has to be compressed such as telecommunication or storage. However, competitive learning has also be used in combination with supervised learning methods, and be applied to function approximation problems or classification problems. We will describe two examples: the "counter propagation" method and the "learning vector quantisation".

14.2.3 Counter Propagation

In a large number of applications, networks that perform vector quantisation are combined with another type of network in order to perform function approximation. An example of such a network is given in

Fig. 14.6. This network can approximate a function $f: \Re^n \rightarrow \Re^m$ by associating with each neuron o a function value $[w_{1o}; w_{2o}, \ldots, w_{mo}]^T$ which is somehow representative for the function values $f(x)$ of inputs x represented by o. This way of approximating a function effectively implements a 'look-up table': an input x is assigned to a table entry k with $\forall_o \neq k$: $||x - w_k|| \leq ||x - w_o||$, and the function value $[w_{1k}; w_{2k}, \ldots, w_{mk}]^T$ in this table entry is taken as an approximation of $f(x)$. A well-known example of such a network is the Counter propagation network (Hecht-Nielsen, 1988).

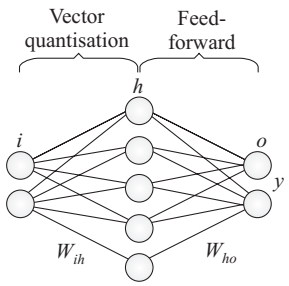

Fig. 14.6 A network combining a vector quantisation layer with a 1-layer feed-forward neural network. This network can be used to approximate functions from $\Re^2$ to $\Re^2$, the input space $\Re^2$ is discretized in 5 disjoint subspaces.

Depending on the application, one can choose to perform the vector quantisation before learning the function approximation, or one can choose to learn the quantisation and the approximation layer simultaneously. As an example of the latter, the network presented in Fig. 14.6 can be supervisedly trained in the following way:

1. Present the network with both input x and function value $d = f(x)$;
2. Perform the unsupervised quantisation step. For each weight vector, calculate the distance from its weight vector to the input pattern and find winner k. Update the weights w_{ih} with equation (14.6);
3. Perform the supervised approximation step:

$$w_{ko}(t+1) = w_{ko}(t) + \gamma(d_o - w_{ko}(t)) \qquad ...(14.13)$$

This is simply the δ – rule with $y_o = \sum_h y_h w_{ho} = w_{ko}$ when k is the winning neuron and the desired output is given by $d = f(x)$.

If we define a function $g(x, k)$ as:

$$g(x, k) = \begin{cases} 1 & \text{if } k \text{ is winner} \\ 0 & \text{otherwise} \end{cases} \qquad ...(14.14)$$

It can be shown that this learning procedure converges to

$$w_{ho} = \int_{\Re^n} y_o g(x, h) dx \qquad ...(14.15)$$

i.e., each table entry converges to the mean function value over all inputs in the subspace represented by that table entry. As we have seen before, the quantisation scheme tracks the input probability density function, which results in a better approximation of the function in those areas where input is most likely to appear.

Not all functions are represented accurately by this combination of quantisation and approximation layers. e.g., a simple identity or combinations of sines and cosines are much better approximated by multilayer back-propagation networks if the activation functions are chosen appropriately. However, if we expect our input to be (a subspace of) a high dimensional input space $<n$ and we expect our function f to be discontinuous at numerous points, the combination of quantisation and approximation is not uncommon and probably very efficient. Of course this combination extends itself much further than the presented combination of the presented single layer competitive learning network and the single layer feed-forward network. The latter could be replaced by a reinforcement learning procedure (see chapter 15).

The quantisation layer can be replaced by various other quantisation schemes, such as Kohonen networks or octree methods (Jansen, Smagt, and Groen, 1994). In fact, various modern statistical function approximation methods (Breiman, Friedman, Olshen, and Stone, 1984; Friedman, 1991) are based on this very idea, extended with the possibility to have the approximation layer influence the quantisation layer (e.g., to obtain a better or locally more fine-grained quantisation).

14.2.4 Learning Vector Quantisation

It is an unpleasant habit in neural network literature, to also cover Learning Vector Quantisation (LVQ) methods in chapters on unsupervised clustering. Granted that these methods also perform a clustering or quantisation task and use similar learning rules, they are trained supervisedly and perform discriminant analysis rather than unsupervised clustering. These networks attempt to define 'decision boundaries' in the input space, given a large set of exemplary decisions (the training set); each decision could, e.g., be a correct class label.

A rather large number of slightly different LVQ methods is appearing in recent literature. They are all based on the following basic algorithm:

1. With each output neuron o, a class label (or decision of some other kind) y_o is associated;
2. A learning sample consists of input vector x^p together with its correct class label y_o^p;
3. Using distance measures between weight vectors w_o and input vector x^p, not only the winner k_1 is determined, but also the second best k_2:
$$||x_p - w_{k1}|| < ||x^p - w_{k_2}|| < ||x^p - w_i|| \; \forall_o \neq k_1, k_2$$
4. The labels $y_{k_1}^p$, $y_{k_2}^p$ are compared with d^p. The weight update rule given in equation (6.6) is used selectively based on this comparison.

An example of the last step is given by the LVQ2 algorithm by Kohonen (1977), using the following strategy:

- if $y_{k_1}^p \neq d^p$ and $d^p = y_{k_2}^p$
- and $||x^p - w_{k_2}|| - ||x^p - w_{k_1}|| < \varepsilon$
- then $w_{k_2}(t+1) = w_{k_2}(t) + \gamma(x - w_{k_2}(t))$
- and $w_{k_1}(t+1) = w_{k_1}(t) + \gamma(x - w_{k_1}(t))$

i.e., w_{k_2} with the correct label is moved towards the input vector, while w_{k_1} with the incorrect label is moved away from it.

The new LVQ algorithms that are emerging all use different implementations of these different steps, e.g., how to define class labels y_o, how many 'next-best' winners are to be determined, how to adapt the number of output neurons i and how to selectively use the weight update rule.

14.3 KOHONEN NETWORK

The Kohonen network (1982, 1984) can be seen as an extension to the competitive learning network, although this is chronologically incorrect. Also, the Kohonen network has a different set of applications.

In the Kohonen network, the output units in S are ordered in some fashion, often in a two-dimensional grid or array, although this is application-dependent. The ordering, which is chosen by the user, determines which output neurons are neighbours.

Now, when learning patterns are presented to the network, the weights to the output units are thus adapted such that the order present in the input space $\Re^2$ is preserved in the output, i.e., the neurons in S. This means that learning patterns which are near to each other in the input space (where 'near' is determined by the distance measure used in finding the winning unit) must be mapped on output units, which are also near to each other, i.e., the same or neighboring units. Thus, if inputs are uniformly distributed in $\Re^N$ and the order must be preserved, the dimensionality of S must be at least N.

The mapping, which represents a discretization of the input space, is said to be topology preserving. However, if the inputs are restricted to a subspace of $\Re^N$, a Kohonen network can be used of lower dimensionality. For example: data on a two- dimensional manifold in a high dimensional input space can be mapped onto a two-dimensional Kohonen network, which can for example be used for visualization of the data.

Usually, the learning patterns are random samples from $\Re^N$. At time t, a sample $x(t)$ is generated and presented to the network. Using the same formulas as in section 6.1, the winning unit k is determined. Next, the weights to this winning unit as well as its neighbours are adapted using the learning rule

$$w_o(t+1) = w_o(t) + \gamma g(o, k)(x(t) - w_o(t)) \qquad ...(14.16)$$

Here, $g(o, k)$ is a decreasing function of the grid-distance between units o and k, such that $g(k, k) = 1$. For example, for $g(\ )$ a Gaussian function can be used, such that (in one dimension!) $g(o, k) = \exp(-(o-k)^2)$ (see Fig. 14.7). Due to this collective learning scheme, input signals, which are near to each other, will be mapped on neighbouring neurons. Thus the topology inherently present in the input signals will be preserved in the mapping, such as depicted in Fig. 14.8.

If the intrinsic dimensionality of S is less than N, the neurons in the network are 'folded' in the input space, such as depicted in Fig. 14.9.

The topology-conserving quality of this network has many counterparts in biological brains. The brain is organized in many places so that aspects of the sensory environment are represented in the form of two-dimensional maps. For example, in the visual system, there are several topographic mappings of visual space onto the surface of the visual cortex. There are organized mappings of the body surface

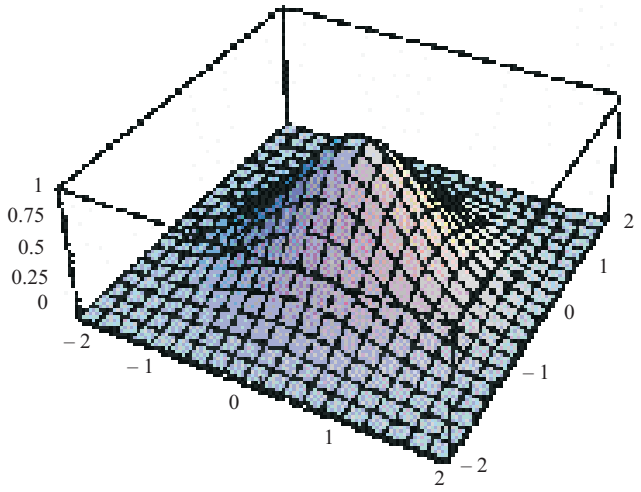

Fig. 14.7 Gaussian neuron distance function g( ). In this case, g( ) is shown for a two-dimensional grid because it looks nice.

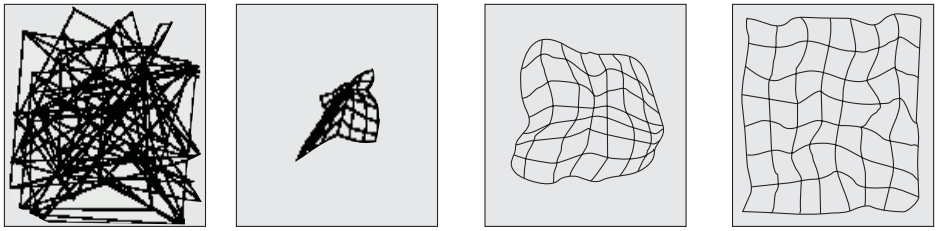

Fig. 14.8 A topology-conserving map converging. The weight vectors of a network with two inputs and 8 x 8 output neurons arranged in a planar grid are shown. A line in each figure connects weight $w_{i,(o_1 o_2)}$ with weights $w_{i,(o_1+1, o_2)}$ and $w_{i,(i_1 i_2+1)}$. The leftmost figure shows the initial weights; the rightmost when the map is almost completely formed.

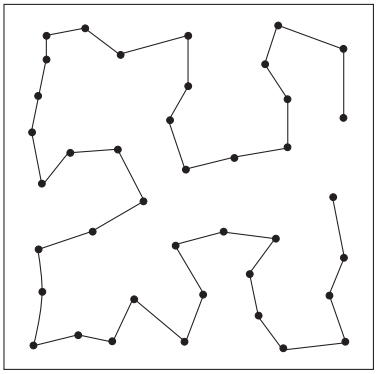

Fig. 14.9 The mapping of a two-dimensional input space on a one-dimensional Kohonen network.

onto the cortex in both motor and somatosensory areas, and tonotopic mappings of frequency in the auditory cortex. The use of topographic representations, where some important aspect of a sensory modality is related to the physical locations of the cells on a surface, is so common that it obviously serves an important information processing function.

It does not come as a surprise, therefore, that already many applications have been devised of the Kohonen topology-conserving maps. Kohonen himself has successfully used the network for phoneme-recognition (Kohonen, Makisara, and Saramaki, 1984). Also, the network has been used to merge sensory data from different kinds of sensors, such as auditory and visual, 'looking' at the same scene (Gielen, Krommenhoek, and Gisbergen, 1991).

To explain the plausibility of a similar structure in biological networks, Kohonen remarks that the lateral inhibition between the neurons could be obtained via efferent connections between those neurons. In one dimension, those connection strengths form a 'Mexican hat' (see Figure 14.10).

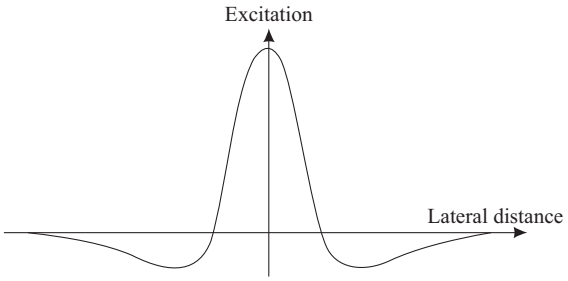

Fig. 14.10 Mexican hat. Lateral interaction around the winning neuron as a function of distance: excitation to nearby neurons, inhibition to farther off neurons.

14.4 PRINCIPAL COMPONENT NETWORKS

The networks presented in the previous sections can be seen as (nonlinear) vector transformations, which map an input vector to a number of binary output elements or neurons. The weights are adjusted in such a way that they could be considered as prototype vectors (vectorial means) for the input patterns for which the competing neuron wins.

The self-organizing transform described in this section rotates the input space in such a way that the values of the output neurons are as uncorrelated as possible and the energy or variances of the patterns is mainly concentrated in a few output neurons. An example is shown in Figure 14.11.

The two dimensional samples (x_1, x_2) are plotted in the figure. It can be easily seen that x_1 and x_2 are related, such that if we know x_1 we can make a reasonable prediction of x_2 and vice versa since the points are centered around the line $x_1 = x_2$. If we rotate the axes over $\pi/4$ we get the (e_1, e_2) axis as plotted in the figure. Here the conditional prediction has no use because the points have uncorrelated coordinates. Another property of this rotation is that the variance or energy of the transformed patterns is maximized on a lower dimension. This can be intuitively verified by comparing the spreads (d_{x_1}, d_{x_2}) and (d_{e_1}, d_{e_2}) in the figures. After the rotation, the variance of the samples is large along the e_1 axis and small along the e_2 axis.

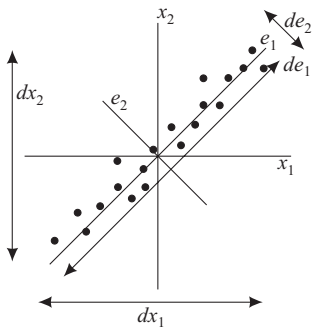

Fig. 14.11 Distribution of input samples.

This transform is very closely related to the eigenvector transformation known from image processing where the image has to be coded or transformed to a lower dimension and reconstructed again by another transform as well as possible.

The next section describes a learning rule which acts as a Hebbian learning rule, but which scales the vector length to unity. In the subsequent section we will see that a linear neuron with a normalised Hebbian learning rule acts as such a transform, extending the theory in the last section to multi-dimensional outputs.

14.4.1 Normalized Hebbian Rule

The model considered here consists of one linear neuron with input weights w. The output $y_o(t)$ of this neuron is given by the usual inner product of its weight w and the input vector x:

$$y_o(t) = w(t)^T x(t) \qquad ...(14.17)$$

As seen in the previous sections, all models are based on a kind of Hebbian learning. However, the basic Hebbian rule would make the weights grow uninhibitedly if there were correlation in the input patterns. This can be overcome by normalising the weight vector to a fixed length, typically 1, which leads to the following learning rule

$$w(t+1) = \frac{w(t) + \gamma y(t) x(t)}{L(w(t) + \gamma y(t) x(t))} \qquad ...(14.18)$$

where $L(\bullet)$ indicates an operator which returns the vector length, and γ is a small learning parameter. Compare this learning rule with the normalized learning rule of competitive learning. There the delta rule was normalized, here the standard Hebb rule is.

Now the operator which computes the vector length, the norm of the vector, can be approximated by a Taylor expansion around $\gamma = 0$:

$$L(w(t)) + \gamma y(t)\, x(t) = 1 + \gamma \left.\frac{\partial L}{\partial \gamma}\right|_{\gamma=0} + O(\gamma^2) \qquad ...(14.19)$$

When we substitute this expression for the vector length in equation (6.18), it resolves for small $\gamma(t^2)$.

$$w(t+1) = (w(t) + \gamma y(t)\, x(t)) \left(1 - \gamma \frac{\partial L}{\partial \gamma}\bigg|_{\gamma=0} + o(\gamma^2) \right) \qquad ...(14.20)$$

Since $\frac{\delta L}{\delta \gamma}\Big|_{\gamma=0} = y(t)^2$ discarding the higher order terms of γ leads to

$$w(t+1) = w(t) + \gamma y(t)\, x(t)(x(t) - y(t) w(t)) \qquad ...(14.21)$$

which is called the 'Oja learning rule' (Oja, 1982). This learning rule thus modifies the weight in the usual Hebbian sense, the first product terms is the Hebb rule $y_o(t)\, x(t)$, but normalizes its weight vector directly by the second product term $-y_o(t)\, x(t)\, w(t)$. What exactly does this learning rule do with the weight vector?

14.4.2 Principal Component Extractor

Remember probability theory? Consider an N-dimensional signal $x(t)$ with

- Mean $\mu = E(x(t))$;
- Correlation matrix $R = E((x(t) - \mu)\, (x(t) - \mu)^T)$.

In the following we assume the signal mean to be zero, so $\mu = 0$.

From equation (6.21) we see that the expectation of the weights for the Oja learning rule equals

$$E(w(t+1)|w(t)) = w(t) + \gamma(Rw(t) - (w(t)^T R w(t)) w(t)) \qquad ...(14.22)$$

which has a continuous counterpart

$$\frac{d}{dt} w(t) = Rw(t) - (w(t)^T R w(t))\, w(t) \qquad ...(14.23)$$

Theorem 14.2: Let the eigenvectors e_i of R be ordered with descending associated eigenvalues λ_i such that $\lambda_1 > \lambda_2 > ... > \lambda_N$. With equation (6.23) the weights $w(t)$ will converge to $\pm e_1$.

Proof: 1 Since the eigenvectors of **R** span the N-dimensional space, the weight vector can be decomposed as

$$w(t) = \sum_i^N \beta_i(t) e_i \qquad ...(14.24)$$

Substituting this in the differential equation and concluding the theorem is left as an exercise.

14.4.3 More Eigenvectors

In the previous section it was shown that a single neuron's weight converges to the eigenvector of the correlation matrix with maximum eigenvalue, i.e., the weight of the neuron is directed in the direction

of highest energy or variance of the input patterns. Here we tackle the question of how to find the remaining eigenvectors of the correlation matrix given the first found eigenvector.

Consider the signal x which can be decomposed into the basis of eigenvectors e_i of its correlation matrix **R**,

$$x = \sum_{i}^{N} \alpha_i e_i \qquad ...(14.25)$$

If we now subtract the component in the direction of e_1, the direction in which the signal has the most energy, from the signal x

$$\tilde{x} = x - \alpha_1 e_1 \qquad ...(14.26)$$

we are sure that when we again decompose ~x into the eigenvector basis, the coefficient $\alpha_1 = 0$, simply because we just subtracted it. We call $\tilde{x}$ the deflation of x.

If now a second neuron is taught on this signal $\tilde{x}$, then its weights will lie in the direction of the remaining eigenvector with the highest eigenvalue. Since the deflation removed the component in the direction of the first eigenvector, the weight will converge to the remaining eigenvector with maximum eigenvalue. In the previous section we ordered the eigenvalues in magnitude, so according to this definition in the limit we will find e_2. We can continue this strategy and find all the N eigenvectors belonging to the signal x.

We can write the deflation in neural network terms if we see that

$$y_o = w^T x = e_1^T \sum_{i}^{N} \alpha_i e_i = \alpha_i \qquad ...(14.27)$$

since

$$w = e_1 \qquad ...(14.28)$$

So that the deflated vector $\tilde{x}$ equals

$$\tilde{x} = x - y_o w \qquad ...(14.29)$$

The term subtracted from the input vector can be interpreted as a kind of a back-projection or expectation. Compare this to ART described in the next section.

14.5 ADAPTIVE RESONANCE THEORY

The last unsupervised learning network we discuss differs from the previous networks in that it is recurrent; as with networks in the next chapter, the data is not only fed forward but also back from output to input units.

14.5.1 Background: Adaptive Resonance Theory

In 1976, Grossberg introduced a model for explaining biological phenomena. The model has three crucial properties:

1. A normalization of the total network activity. Biological systems are usually very adaptive to large changes in their environment. For example, the human eye can adapt itself to large variations in light intensities;
2. Contrast enhancement of input patterns. The awareness of subtle differences in input patterns can mean a lot in terms of survival. Distinguishing a hiding panther from a resting one makes all the difference in the world. The mechanism used here is contrast enhancement.
3. Short-term memory (STM) storage of the contrast-enhanced pattern. Before the input pattern can be decoded, it must be stored in the short-term memory. The long-term memory (LTM) implements an arousal mechanism (i.e., the classification), whereas the STM is used to cause gradual changes in the LTM.

The system consists of two layers, F_1 and F_2, which are connected to each other via the LTM (see

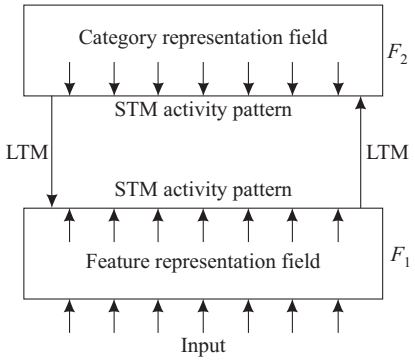

Fig. 14.12 The ART architecture.

Fig. 14.12). The input pattern is received at F_1, whereas classification takes place in F_2. As mentioned before, the input is not directly classified. First a characterization takes place by means of extracting features, giving rise to activation in the feature representation field.

The expectations, residing in the LTM connections, translate the input pattern to a categorization in the category representation field. The classification is compared to the expectation of the network, which resides in the LTM weights from F_2 to F_1. If there is a match, the expectations are strengthened otherwise the classification is rejected.

14.5.2 ART1: The Simplified Neural Network Model

The ART1 simplified model consists of two layers of binary neurons (with values 1 and 0), called F_1 (the comparison layer) and F_2 (the recognition layer) (see Fig. 14.13). Each neuron in F_1 is connected to all neurons in F_2 via the continuous-valued forward long term memory (LTM) W^f, and vice versa via the binary-valued backward LTM W^b. The other modules are gain 1 and 2 (G_1 and G_2), and a reset module.

Each neuron in the comparison layer receives three inputs: a component of the input pattern, a component of the feedback pattern, and a gain G_1. A neuron outputs a 1 if and only if at least three of these inputs are high: the ‘two-thirds rule’.

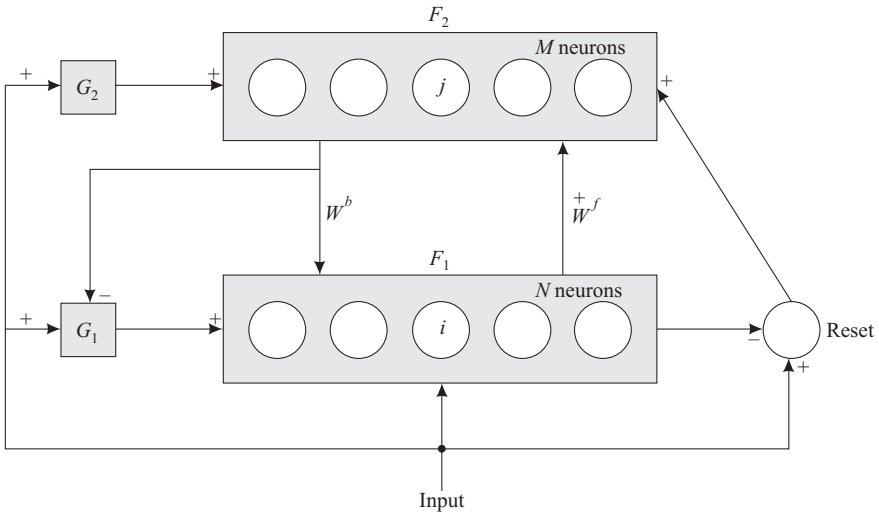

Fig. 14.13 The ART 1 neural network.

The neurons in the recognition layer each compute the inner product of their incoming (continuous-valued) weights and the pattern sent over these connections. The winning neuron then inhibits all the other neurons via lateral inhibition.

Gain 2 is the logical 'or' of all the elements in the input pattern x.

Gain 1 equals gain 2, except when the feedback pattern from F_2 contains any 1; then it is forced to zero.

Finally, the reset signal is sent to the active neuron in F_2 if the input vector x and the output of F_1 differ by more than some vigilance level.

14.5.3 Operation

The pattern is sent to F_2, and in F_2 one neuron becomes active. This signal is then sent back over the backward LTM, which reproduces a binary pattern at F_1. Gain 1 is inhibited, and only the neurons in F_1 which receive a 'one' from both x and F_2 remain active.

If there is a substantial mismatch between the two patterns, the reset signal will inhibit the neuron in F_2 and the process is repeated.

Instead of following Carpenter and Grossberg's description of the system using differential equations, we use the notation employed by Lippmann (1987):

1. Initialization:

$$w_{ji}^{b}(0) = 1$$

$$w_{ji}^{f} = \frac{1}{1+N}$$

where N is the number of neurons in F_1, M the number of neurons in F_2, $0 \le i < N$, and $0 \le j < M$. Also, choose the vigilance threshold ρ, $0 \le \rho \le 1$;

2. Apply the new input pattern x;
3. Compute the activation values of the neurons in F_2:

$$y_i' = \sum_{j=1}^{N} w_{ij}^f(t) x_1 \qquad ...(14.30)$$

4. Select the winning neuron $k (0 \le k < M)$;
5. Vigilance test: if

$$\frac{w_k^b(t) \circ x}{x \circ x} > \rho \qquad ...(14.31)$$

where o denotes inner product, go to step 7, else go to step 6. Note that $w_k^b \circ x$ essentially is the inner product $x^* \circ x$, which will be large if x^* and x are near to each other;

6. Neuron k is disabled from further activity. Go to step 3;
7. Set for all l, $o \le l < N$:

$$w_{kl}^b(t+1) = w_{kl}^b(t) x_l$$

$$w_{lk}^f(t+1) = \frac{w_{kl}^b(t) x_l}{\frac{1}{2} + \sum_{i=1}^{N} w_{ki}^b(t) x_i}$$

8. Re-enable all neurons in F_2 and go to step 2.

Fig. 14.14 shows exemplar behaviour of the network.

14.5.4 ART 1: The Original Model

In later work, Carpenter and Grossberg (1987) present several neural network models to incorporate parts of the complete theory. We will only discuss the first model, ART 1.

The network incorporates a follow-the-leader clustering algorithm (Hartigan, 1975). This algorithm tries to fit each new input pattern in an existing class. If no matching class can be found, i.e., the distance between the new pattern and all existing classes exceeds some threshold, a new class is created containing the new pattern.

The novelty in this approach is that the network is able to adapt to new incoming patterns, while the previous memory is not corrupted. In most neural networks, such as the back- propagation network, all patterns must be taught sequentially; the teaching of a new pattern might corrupt the weights for all previously learned patterns. By changing the structure of the network rather than the weights, ART1 overcomes this problem.

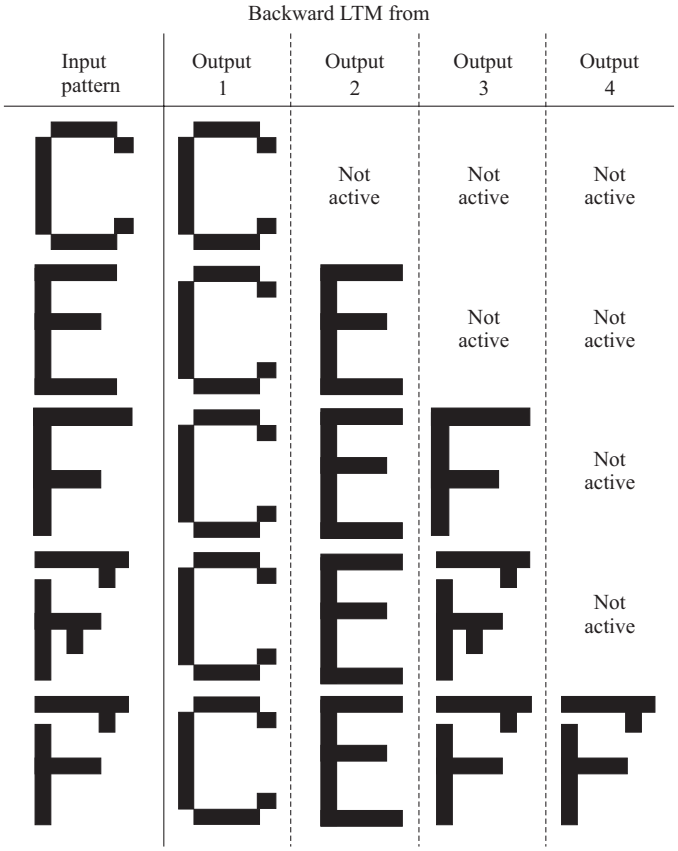

Fig. 14.14 An example of the behaviour of the Carpenter Grossberg network for letter patterns. The binary input patterns on the left were applied sequentially. On the right the stored patterns (i.e., the weights of W^b for the first four output units) are shown.

14.5.5 Normalization of the Original Model

We will refer to a cell in F_1 or F_2 with k.

Each cell k in F_1 or F_2 receives an input s_k and respond with an activation level y_k.

In order to introduce normalization in the model, we set $I = \sum s_k$ and let the relative input intensity $\Theta_k = s_k I^{-1}$.

So we have a model in which the change of the response y_k of an input at a certain cell k

- depends inhibitorily on all other inputs and the sensitivity of the cell, i.e., the surroundings of each cell have a negative influence on the cell $-y_k \sum_{l \neq k} s_l$;
- has an excitatory response as far as the input at the cell is concerned $+Bs_k$;

- has an inhibitory response for normalization $-y_k s_k$;
- has a decay $-Ay_k$.

Here, A and B are constants. The differential equation for the neurons in F_1 and F_2 now is

$$\frac{dy_k}{dt} = -Ay_k + (B - yk)s_k - y_k \sum_{l \neq k} s_l \qquad ...(14.32)$$

with $0 \leq y_k(0) \leq B$ because the inhibitory effect of an input can never exceed the excitatory input.

At equilibrium, when $\frac{dy_k}{dt} = 0$, and with $I = \sum s_k$ we have that

$$y_k(A + 1) = Bs_k \qquad ...(14.33)$$

Because of the definition of $\Theta_k = s_k I^{-1}$ we get

$$y_k = \Theta_k \frac{BI}{A + I} \qquad ...(14.34)$$

Therefore, at equilibrium y_k is proportional to Θ_k, and, since

$$\frac{BI}{A + I} \leq B \qquad ...(14.35)$$

The total activity $y^{\text{total}} = \sum y_k$ never exceeds B: it is normalized.

14.5.6 Contrast Enhancement

In order to make F_2 react better on differences in neuron values in F_1 (or vice versa), contrast enhancement is applied: the contrasts between the neuronal values in a layer are amplified. We can show that eq. (14.32) does not suffice anymore. In order to enhance the contrasts, we chop off all the equal fractions (uniform parts) in F_1 or F_2. This can be done by adding an extra inhibitory input proportional to the inputs from the other cells with a factor C:

$$\frac{dy_k}{dt} = -Ay_k + (B - y_k)s_k - (y_k + C) \sum_{l \neq k} s_l \qquad ...(14.36)$$

At equilibrium, when we set $B = (n - 1)\, C$ where n is the number of neurons, we have

$$y_k = \frac{nCI}{A + I}\left(\Theta_k - \frac{1}{n}\right) \qquad ...(14.37)$$

Now, when an input in which all the s_k are equal is given, then all the y_k are zero: the effect of C is enhancing differences. If we set $B \leq (n - 1)C$ or $C/(B + C) \geq 1/n$, then more of the input shall be chopped off.

The description of ART1 continues by defining the differential equations for the LTM. Instead of following Carpenter and Grossberg's description, we will revert to the simplified model as presented by Lippmann.

QUESTION BANK

1. What are the advantages of self-organizing networks?
2. What is competitive learning network?
3. Explain various methods of determining the winner and the corresponding learning rule.
4. Describe the square error criterion to measure the quality of a given clustering.
5. Describe the vector quantisation scheme.
6. Explain the counter propagation network.
7. Describe the learning vector quantisation method.
8. What is Kohonen network? Explain.
9. Explain normalized Hebbian rule.
10. Describe adaptive resonance theory.
11. Explain ART 1 neural network.
12. Describe the normalization of ART 1.

REFERENCES

1. D.E. Rumelhart, and D. Zipser, Feature discovery by competitive learning. *Cognitive Science*, Vol. 9, pp. 75-112, 1985.
2. G.A. Carpenter, and S. Grossberg, A massively parallel architecture for a self-organizing neural pattern recognition machine, *Computer Vision*, *Graphics*, *and Image Processing*, Vol. 37, pp. 54-115, 1987.
3. S. Grossberg, Adaptive pattern classification and universal recoding I & II, *Biological Cybernetics*, Vol. 23, pp. 121-134, 187-202, 1976.
4. K. Fukushima, Cognitron: A self-organizing multilayered neural network, *Biological Cybernetics*, Vol. 20, pp. 121-136, 1975.
5. K. Fukushima, Neocognitron: A hierarchical neural network capable of visual pattern recognition, *Neural Networks*, Vol. 1, pp. 119-130, 1988.
6. R.P. Lippmann, Review of neural networks for speech recognition, *Neural Computation*,Vol. 1, pp. 1-38, 1989.
7. S.C. Ahalt, A.K. Krishnamurthy, P. Chen, D. Melton, Competitive learning algorithms for vector quantisation, *Neural Networks*, Vol. 3, pp. 277-290, 1990.
8. R.H. Nielsen, Counterpropagation networks, *Neural Networks*, Vol. 1, No. 131-139, 1988.
9. A. Jansen, P.P. Smagt, P. Van der, and F.C.A. Groen, Nested networks for robot control, In A.F. Murray (Ed.), *Neural Network Applications*, Kluwer Academic Publishers, 1994.
10. L. Breiman, J.H. Friedman, R.A. Olshen, and C.J. Stone, *Classification and Regression Trees*. Wadsworth and Broks/Cole, 1984.
11. J.H. Friedman, Multivariate adaptive regression splines, *Annals of Statistics*, Vol. 19, pp. 1-141, 1991.

12. T. Kohonen, *Associative Memory*: *A System-Theoretical Approach*, Springer-Verlag, 1977.
13. T. Kohonen, Self-organized formation of topologically correct feature maps, Biological Cybernetics, 43, 59-69, 1982.
14. T. Kohonen, *Self-Organization and Associative Memory*, Berlin: Springer-Verlag, 1984.
15. T. Kohonen, M. Makisara, and T. Saramaki, Phonotopic maps|insightful representation of phonological features for speech recognition, *In Proceedings of the 7th IEEE International Conference on Pattern Recognition*, DUNNO, 1984.
16. C. Gielen, K. Krommenhoek, and J. Gisbergen, A procedure for self-organized sensor-fusion in topologically ordered maps, In T. Kanade, F.C.A. Groen, and L.O. Hertzberger (Eds.), *Proceedings of the Second International Conference on Autonomous Systems*, Elsevier Science Publishers, pp. 417-423, 1991.
17. E. Oja, A simplified neuron model as a principal component analyzer, *Journal of Mathematical Biology*, Vol. 15, pp. 267-273, 1982.
18. S. Grossberg, Adaptive pattern classification and universal recoding I & II, *Biological Cybernetics*, Vol. 23, 121-134, 187-202, 1976.
19. R.P. Lippmann, An introduction to computing with neural nets, *IEEE Transactions on Acoustics*, *Speech*, *and Signal Processing*, Vol. 2 No.4, pp. 4-22, 1987.
20. G.A. Carpenter, and S. Grossberg, A massively parallel architecture for a self-organizing neural pattern recognition machine, *Computer Vision*, *Graphics*, *and Image Processing*, Vol. 37, pp. 54-115, 1987.
21. G.A. Carpenter, and S. Grossberg, ART 2: Self-organization of stable category recognition codes for analog input patterns, *Applied Optics*, 26(23), 4919-4930, 1987.
22. J.A. Hartigan, *Clustering Algorithms*, New York: John Wiley & Sons, 1975.

CHAPTER 15

Reinforcement Learning

15.1 INTRODUCTION

In the previous chapters a number of supervised training methods have been described in which the weight adjustments are calculated using a set of 'learning samples', existing of input and desired output values. However, not always such a set of learning examples is available. Often the only information is a scalar evaluation r, which indicates how well the neural network is performing. Reinforcement learning involves two subproblems. The First is that the 'reinforcement' signal r is often delayed since it is a result of network outputs in the past. This temporal credit assignment problem is solved by learning a 'critic' network which represents a cost function J predicting future reinforcement. The second problem is to Find a learning procedure which adapts the weights of the neural network such that a mapping is established which minimizes J. The two problems are discussed in the next paragraphs, respectively. Fig. 15.1 shows a reinforcement-learning network interacting with a system.

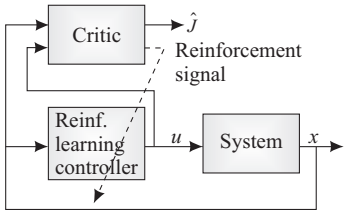

Fig. 15.1 Reinforcement learning scheme.

15.2 THE CRITIC

The first problem is how to construct a critic, which is able to evaluate system performance. If the objective of the network is to minimize a direct measurable quantity r, performance feedback is straightforward and a critic is not required. On the other hand, how is current behaviour to be evaluated

if the objective concerns future system performance? The performance may for instance be measured by the cumulative or future error. Most reinforcement learning methods (Barto, Sutton and Anderson (1983) use the temporal difference (TD) algorithm (Sutton, 1988) to train the critic.

Suppose the immediate cost of the system at time step k are measured by $r(x_k, u_k, k)$, as a function of system states x_k and control actions (network outputs) u_k. The immediate measure r is often called the external reinforcement signal in contrast to the internal reinforcement signal in Fig. 7.1. Define the performance measure $J(x_k, u_k, k)$of the system as a discounted cumulative of future cost. The task of the critic is to predict the performance measure:

$$J(x_k, u_k, k) = \sum_{i=k}^{\infty} y^{i-k} r(x_k, u_k, k) \qquad ...(15.1)$$

in which $\gamma \in [0, 1]$ is a discount factor (usually ≈ 0.95).

The relation between two successive prediction can easily be derived:

$$J(x_k, u_k, k) = r(x_k, u_k, k) + rJ(x_{k+1}, u_{k+1}, k+1) \qquad ...(15.2)$$

If the network is correctly trained, the relation between two successive network outputs $\hat{J}$ should be:

$$\hat{J}(x_k, u_k, k) = r(x_k, u_k, k) + r\hat{J}(x_{k+1}, u_{k+1}, k+1) \qquad ...(15.3)$$

If the network is not correctly trained, the temporal difference $\delta(k)$ between two successive predictions is used to adapt the critic network:

$$\delta(k) = [r(x_k, u_k, k) + r\hat{J}(x_{k+1}, u_{k+1}, k+1)] - \hat{J}(x_k, u_k, k) \qquad ...(15.4)$$

A learning rule for the weights of the critic network $w_c(k)$, based on minimizing $\delta^2(k)$ can be derived:

$$\Delta w_c(k) = -\alpha\varepsilon(k)\frac{\partial \hat{J}(x_k, u_k, k)}{\partial w_c(k)} \qquad ...(15.5)$$

in which α is the learning rate.

15.3 THE CONTROLLER NETWORK

If the critic is capable of providing an immediate evaluation of performance, the controller network can be adapted such that the optimal relation between system states and control actions is found. Three approaches are distinguished:

1. In case of a finite set of actions U, all actions may virtually be executed. The action which decreases the performance criterion most is selected:

$$u_k = \min_{u \in U} \hat{J}(x_k, u_k, k) \qquad ...(15.6)$$

The RL-method with this 'controller' is called Q-learning (Watkins & Dayan, 1992). The method approximates dynamic programming which will be discussed in the next section.

2. If the performance measure $J(x_k, u_k, k)$ is accurately predicted, then the gradient with respect to the controller command u_k can be calculated, assuming that the critic network is differentiable. If the measure is to be minimized, the weights of the controller w_r are adjusted in the direction of the negative gradient:

$$\Delta w_r(k) = -\beta \frac{\partial \hat{J}(x_k, u_k\, k)}{\partial u(k)} \frac{\partial u(k)}{\partial w_r(k)} \qquad ...(15.7)$$

with β being the learning rate. Werbos (1992) has discussed some of these gradient based algorithms in detail. Sofge and White (1992) applied one of the gradient based methods to optimize a manufacturing process.

3. A direct approach to adapt the controller is to use the difference between the predicted and the 'true' performance measure as expressed in equation 15.3. Suppose that the performance measure is to be minimized. Control actions that result in negative differences, i.e. the true performance is better than was expected, then the controller has to be 'rewarded'. On the other hand, in case of a positive difference, then the control action has to be 'penalized'. The idea is to explore the set of possible actions during learning and incorporate the beneficial ones into the controller. Learning in this way is related to trial-and-error learning studied by psychologists in which behavior is selected according to its consequences.

   Generally, the algorithms select probabilistically actions from a set of possible actions and update action probabilities on basis of the evaluation feedback. Most of the algorithms are based on a look-up table representation of the mapping from system states to actions (Barto et al., 1983). Each table entry has to learn which control action is best when that entry is accessed. It may be also possible to use a parametric mapping from systems states to action probabilities. Gullapalli (1990) adapted the weights of a single layer network.

15.4 BARTO'S APPROACH: THE ASE-ACE COMBINATION

Barto, Sutton and Anderson (1983) have formulated 'reinforcement learning' as a learning strategy, which does not need a set of examples provided by a 'teacher'. The system described by Barto explores the space of alternative input-output mappings and uses an evaluative feedback (reinforcement signal) on the consequences of the control signal (network output) on the environment. It has been shown that such reinforcement learning algorithms are implementing an on-line, incremental approximation to the dynamic programming method for optimal control, and are also called 'heuristic' dynamic programming (Werbos, 1990).

The basic building blocks in the Barto network are an Associative Search Element (ASE) which uses a stochastic method to determine the correct relation between input and output and an Adaptive Critic Element (ACE) which learns to give a correct prediction of future reward or punishment (Fig. 15.2). The external reinforcement signal r can be generated by a special sensor (for example a collision sensor of a mobile robot) or be derived from the state vector. For example, in control applications, where the state s of a system should remain in a certain part A of the control space, reinforcement is given by:

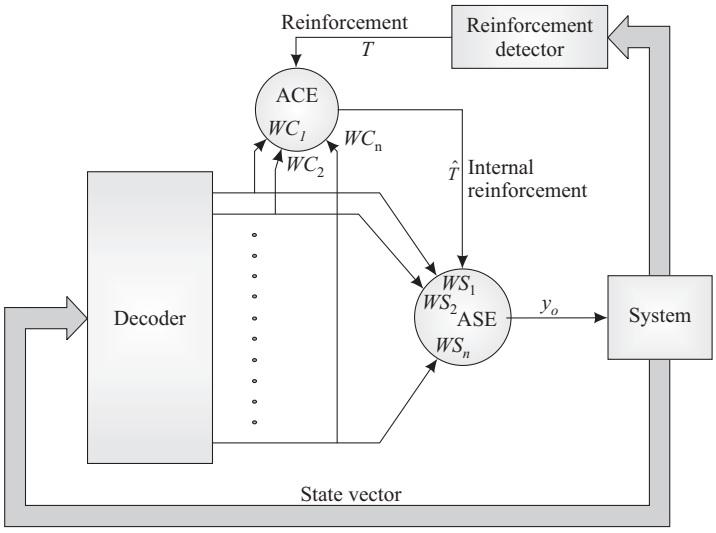

Fig. 15.2: Architecture of a reinforcement learning scheme with critic element.

$$r = \begin{cases} 0 & \text{if } s \in A \\ -1 & \text{otherwise} \end{cases} \qquad ...(15.8)$$

15.4.1 Associative Search

In its most elementary form the ASE gives a binary output value $y_o(t) \in \{0, 1\}$; as a stochastic function of an input vector. The total input of the ASE is, similar to the neuron presented in chapter 2, the weighted sum of the inputs, with the exception that the bias input in this case is a stochastic variable N with mean zero normal distribution:

$$s(t) = \sum_{j=1}^{N} w_{sj} x_j (t) + N_j \qquad ...(15.9)$$

The activation function F is a threshold such that

$$y_o(t) = y(t) \begin{cases} 1 & \text{if } s(t) > 0 \\ 0 & \text{otherwise} \end{cases} \qquad ...(15.10)$$

For updating the weights, a Hebbian type of learning rule is used. However, the update is weighted with the reinforcement signal $r(t)$ and an 'eligibility' e_j is defined instead of the product $y_0(t)\ x_j(t)$ of input and output:

$$w_{sj}(t + 1) = w_{sj}(t) + \alpha r(t)\ e_j(t) \qquad ... (15.11)$$

where α is a learning factor. The eligibility e_j is given by

$$e_j(t+1) = \delta e_j(t) + (1-\delta)\, y_0(t)\, x_j(t) \qquad ...(15.12)$$

with δ the decay rate of the eligibility. The eligibility is a sort of 'memory'; e_j is high if the signals from the input state unit j and the output unit are correlated over some time.

Using $r(t)$ in expression (15.11) has the disadvantage that learning only finds place when there is an external reinforcement signal. Instead of $r(t)$, usually a continuous internal reinforcement signal $\hat{r}(t)$ given by the ACE, is used.

Barto and Anandan (1985) proved convergence for the case of a single binary output unit and a set of linearly independent patterns x^p. In control applications, the input vector is the (n-dimensional) state vector s of the system. In order to obtain a linear independent set of patterns x^p, often a 'decoder' is used, which divides the range of each of the input variables s_i in a number of intervals. The aim is to divide the input (state) space in a number of disjunct subspaces or boxes. The input vector can therefore only be in one subspace at a time. The decoder converts the input vector into a binary valued vector x, with only one element equal to one, indicating which subspace is currently visited. It has been shown (Krose and Dam, 1992) that instead of a-priori quantisation of the input space, a self-organizing quantisation, based on methods described in this chapter, results in a better performance.

15.4.2 Adaptive Critic

The Adaptive Critic Element (ACE, or 'evaluation network') is basically the same as described in section 7.1. An error signal is derived from the temporal difference of two successive predictions (in this case denoted by p!) and is used for training the ACE:

$$\hat{r}(t) = r(t) + \gamma p(t) - p(t-1) \qquad ...(15.13)$$

$p(t)$ is implemented as a series of 'weights' w_{Ck} to the ACE such that

$$p(t) = w_{Ck} \qquad ...(15.14)$$

if the system is in state k at time t, denoted by $x_k = 1$. The function is learned by adjusting the w_{Ck}'s according to a 'delta-rule' with an error signal δ given by $\hat{r}(t)$:

$$\Delta w_{Ck}(t) = \beta \hat{r}(t) h_j(t) \qquad ...(15.15)$$

β is the learning parameter and $h_j(t)$ indicates the 'trace' of neuron x_j:

$$h_j(t) = \lambda h_j(t-1) + (1-\lambda)\, x_j(t-1) \qquad ...(15.16)$$

This trace is a low-pass filter or momentum, through which the credit assigned to state j increases while state j is active and decays exponentially after the activity of j has expired. If $\hat{r}(t)$ is positive, the action u of the system has resulted in a higher evaluation value, whereas a negative $\hat{r}(t)$ indicates a deterioration of the system. $\hat{r}(t)$ can be considered as an internal reinforcement signal.

15.4.3 The Cart-Pole System

An example of such a system is the cart-pole balancing system (see Fig. 15.3). Here, a dynamics controller must control the cart in such a way that the pole always stands up straight. The controller applies a 'left' or 'right' force F of fixed magnitude to the cart, which may change direction at discrete time intervals. The model has four state variables:

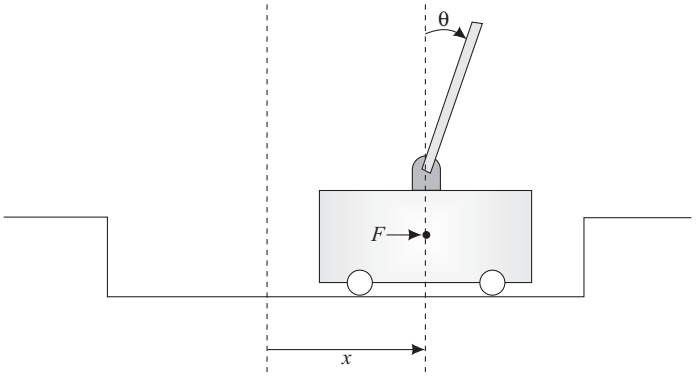

Fig. 15.3: The cart-pole system.

x the position of the cart on the track,

θ the angle of the pole with the vertical,

$\dot{x}$ the cart velocity, and

$\dot{\theta}$ the angle velocity of the pole.

Furthermore, a set of parameters specify the pole length and mass, cart mass, coefficients of friction between the cart and the track and at the hinge between the pole and the cart, the control force magnitude, and the force due to gravity. The state space is partitioned on the basis of the following quantisation thresholds:

1. x : $\pm 0.8, \pm 2.4$ m
2. θ : $0°, \pm 1°, \pm 6°, \pm 12°$
3. $\dot{x}$: $\pm 0.5, \pm\infty$ m/s
4. $\dot{\theta}$: $\pm 50, \pm\infty$ °/s

This yields $3 \times 6 \times 3 \times 3 = 162$ regions corresponding to all of the combinations of the intervals. The decoder output is a 162-dimensional vector. A negative reinforcement signal is provided when the state vector gets out of the admissible range:
when $x > 2.4$, $x < -2.4$, $\theta > 12°$ or $\theta < -12°$.

The system has proved to solve the problem in about 75 learning steps.

15.5 REINFORCEMENT LEARNING VERSUS OPTIMAL CONTROL

The objective of optimal control is to generate control actions in order to optimize a predefined performance measure. One technique to find such a sequence of control actions which define an optimal control policy is Dynamic Programming (DP). The method is based on the principle of optimality, formulated by Bellman (1957): Whatever the initial system state, if the first control action is contained

in an optimal control policy, then the remaining control actions must constitute an optimal control policy for the problem with as initial system state the state remaining from the first control action. The 'Bellman equations' follow directly from the principle of optimality. Solving the equations backwards in time is called dynamic programming.

Assume that a performance measure $J(x_k, u_k, k) = \sum_{i=k}^{N} J(x_i, u_i, k)$ with r being the immediate costs, is to be minimized. The minimum costs $J_{\min}$ of cost J can be derived by the Bellman equations of DP. The equations for the discrete case are (White & Jordan, 1992):

$$J_{\min}(x_k, u_k, k) = \min_{u \in U} [J_{\min} x_{k+1}, u_{k+1}, k + 1) + r(x_k, u_k, k)] \quad ...(15.17)$$

$$J_{\min}(x_N) = r(x_N) \quad ...(15.18)$$

The strategy for finding the optimal control actions is solving equation (15.17) and (15.18) from which u_k can be derived. This can be achieved backwards, starting at state x_N. The requirements are a bounded N, and a model, which is assumed to be an exact representation of the system and the environment. The model has to provide the relation between successive system states resulting from system dynamics, control actions and disturbances. In practice, a solution can be derived only for a small N and simple systems. In order to deal with large or infinity N, the performance measure could be defined as a discounted sum of future costs as expressed by equation 15.2.

Reinforcement learning provides a solution for the problem stated above without the use of a model of the system and environment. RL is therefore often called an 'heuristic' dynamic programming technique (Barto, Sutton, & Watkins, 1990),(Sutton, Barto, & Wilson, 1992), (Werbos, 1992). The most directly related RL-technique to DP is Q-learning (Watkins & Dayan, 1992). The basic idea in Q-learning is to estimate a function, Q, of states and actions, where Q is the minimum discounted sum of future costs $J_{\min}(x_k, u_k, k)$ (the name 'Q-learning' comes from Watkins' notation). For convenience, the notation with J is continued here:

$$\hat{J}(x_k, u_k, k) = \gamma J_{\min}(x_{k+1}, u_{k+1}, k + 1) + r(x_k, u_k, k) \quad ...(15.19)$$

The optimal control rule can be expressed in terms of $\hat{J}$ by noting that an optimal control action for state x_k is any action u_k that minimizes $\hat{J}$ according to equation 7.6.

The estimate of minimum cost $\hat{J}$ is updated at time step $k + 1$ according equation 7.5. The temporal difference $\varepsilon(k)$between the 'true' and expected performance is again used:

$$\varepsilon(k) = [\gamma \min_{u \in U} \hat{J}(x_{k+1}, u_{k+1}, k + 1) + r(x_k, u_k, k)] - \hat{J}(x_k, u_k, k) \quad ...(15.20)$$

Watkins has shown that the function converges under some pre-specified conditions to the true optimal Bellmann equation (Watkins & Dayan, 1992): (1) the critic is implemented as a look-up table; (2) the learning parameter α must converge to zero; (3) all actions continue to be tried from all states.

QUESTION BANK

1. Explain reinforcement learning scheme.
2. What are the various approaches of control networks used to find optimal relation between system states and control actions?
3. Describe the Barto network of reinforcement learning.
4. What are the building blocks of Barto network? Explain them.
5. Explain the cast-pole balancing scheme.
6. Describe dynamic programming to find a sequence of acontrol actions.

REFERENCES

1. A.G. Barto, R.S. Sutton, and C.W. Anderson, Neuron like adaptive elements that can solve difficult learning problems, IEEE Transactions on Systems, *Man and Cybernetics*, Vol. 13, pp. 834-846, 1983.
2. R.S. Sutton, Learning to predict by the methods of temporal differences, *Machine Learning*, Vol. 3, pp. 9-44, 1988.
3. C.J. C.H. Watkins, and P. Dayan, Q-learning, *Machine Learning*, Vol. 8, pp. 279-292, 1992.
4. P. Werbos, Approximate dynamic programming for real-time control and neural modeling, In D. Sofge & D. White (Eds.), *Handbook of Intelligent Control*, *Neural*, *Fuzzy*, *and Adaptive Approaches*. Van Nostrand Reinhold, New York, 1992.
5. D. Sofge, and D. White, Applied learning: optimal control for manufacturing, In D. Sofge & D. White (Eds.), *Handbook of Intelligent Control, Neural, Fuzzy, and Adaptive Approaches*, Van Nostrand Reinhold, New York, 1992.
6. A.G. Barto, R.S. Sutton, C.W. Anderson, Neuronlike adaptive elements that can solve difficult learning problems, *IEEE Transactions on Systems*, *Man and Cybernetics*, Vol. 13, pp. 834-846, 1983.
7. V. Gullapalli, A stochastic reinforcement learning algorithm for learning real-valued functions, *Neural Networks*, Vol. 3, pp. 671-692, 1990.
8. P.W. Werbos, A menu for designs of reinforcement learning over time. In W.T.M. III, R.S. Sutton, & P.J. Werbos (Eds.), *Neural Networks for Control*, MIT Press/Bradford, 1990.
9. A.G. Barto, and P. Anandan, Pattern-recognizing stochastic learning automata, *IEEE Transactions on Systems*, *Man and Cybernetics*, Vol. 15, pp. 360-375, 1985.
10. B.J.A. Krose, and J.W.M. Dam, Learning to avoid collisions: A reinforcement learning paradigm for mobile robot manipulation, In Proceedings of IFAC/IFIP/IMACS International Symposium on Artificial Intelligence in Real-Time Control, Delft: IFAC, Luxemburg, pp. 295-300 1992.
11. R. Bellman, *Dynamic Programming*, Princeton University Press, 1957.
12. D. White, and M. Jordan, Optimal control: a foundation for intelligent control, In D. Sofge & D. White (Eds.), *Handbook of Intelligent Control*, *Neural*, *Fuzzy*, *and Adaptive Approaches*, Van Nostrand Reinhold, New York, 1992.

13. A.G. Barto, R.S. Sutton, and C. Watkins, Sequential decision problems and neural networks. In D. Touretsky (Ed.), Advances in *Neural Information Processing* II, DUNNO, 1990.
14. R.S. Sutton, A. Barto, and R. Wilson, Reinforcement learning is direct adaptive optimal control, *IEEE Control Systems*, Vol. 6, pp. 19-22, 1992.
15. P. Werbos, Approximate dynamic programming for real-time control and neural modeling, In D. Sofge & D. White (Eds.), *Handbook of Intelligent Control*, *Neural*, *Fuzzy*, *and Adaptive Approaches*. Van Nostrand Reinhold, New York, 1992.
16. C.J. C.H. Watkins, and P. Dayan, *Q-learning*, *Machine Learning*, Vol. 8, pp. 279-292, 1992.

CHAPTER 16

Neural Networks Applications

16.1 INTRODUCTION

A list of some applications mentioned in the literature follows:

1. **Aerospace:** High performance aircraft autopilot, flight path simulation, aircraft control systems, autopilot enhancements, aircraft component simulation, aircraft component fault detection.
2. **Automotive:** Automobile automatic guidance system, warranty activity analysis.
3. **Banking:** Check and other document reading, credit application evaluation.
4. **Credit Card Activity Checking:** Neural networks are used to spot unusual credit card activity that might possibly be associated with loss of a credit card.
5. **Defense:** Weapon steering, target tracking, object discrimination, facial recognition, new kinds of sensors, sonar, radar and image signal processing including data compression, feature extraction and noise suppression, signal/image identification.
6. **Electronics:** Code sequence prediction, integrated circuit chip layout, process control, chip failure analysis, machine vision, voice synthesis, nonlinear modeling.
7. **Entertainment:** Animation, special effects, market forecasting.
8. **Financial:** Real estate appraisal, loan advisor, mortgage screening, corporate bond rating, credit-line use analysis, portfolio trading program, corporate financial analysis, currency price prediction.
9. **Industrial:** Neural networks are being trained to predict the output gasses of furnaces and other industrial processes. They then replace complex and costly equipment used for this purpose in the past.
10. **Insurance:** Policy application evaluation, product optimization.
11. **Manufacturing:** Manufacturing process control, product design and analysis, process and machine diagnosis, real-time particle identification, visual quality inspection systems, beer testing, welding quality analysis, paper quality prediction, computer-chip quality analysis, analysis of grinding operations, chemical product design analysis, machine maintenance analysis, project bidding, planning and management, dynamic modeling of chemical process system.

12. **Medical:** Breast cancer cell analysis, EEG and ECG analysis, prosthesis design, optimization of transplant times, hospital expense reduction, hospital quality improvement, emergency-room test advisement.
13. **Oil and Gas:** Exploration.
14. **Robotics:** Trajectory control, forklift robot, manipulator controllers, vision systems.
15. **Speech:** Speech recognition, speech compression, vowel classification, text-to-speech synthesis.
16. **Securities:** Market analysis, automatic bond rating, stock trading advisory systems.
17. **Telecommunications:** Image and data compression, automated information services, real-time translation of spoken language, customer payment processing systems.
18. **Transportation:** Truck brake diagnosis systems, vehicle scheduling, routing systems.

16.2 ROBOT CONTROL

An important area of application of neural networks is in the field of robotics. Usually, these networks are designed to direct a manipulator, which is the most important form of the industrial robot, to grasp objects, based on sensor data. Another applications include the steering and path-planning of autonomous robot vehicles.

In robotics, the major task involves making movements dependent on sensor data. There are four related problems to be distinguished (Craig, 1989):

- Forward kinematics
- Inverse kinematics
- Dynamics
- Trajectory generation

16.2.1 Forward Kinematics

Kinematics is the science of motion, which treats motion without regard to the forces, which cause it. Within this science one studies the position, velocity, acceleration, and all higher order derivatives of the position variables. A very basic problem in the study of mechanical manipulation is that of forward kinematics. This is the static geometrical problem of computing the position and orientation of the end-effector (‘hand’) of the manipulator. Specifically, given a set of joint angles, the forward kinematic problem is to compute the position and orientation of the tool frame relative to the base frame (see Fig. 16.1).

16.2.2 Inverse Kinematics

This problem is posed as follows: given the position and orientation of the end-effector of the manipulator, calculate all possible sets of joint angles which could be used to attain this given position and orientation. This is a fundamental problem in the practical use of manipulators.

The inverse kinematic problem is not as simple as the forward one. Because the kinematic equations are nonlinear, their solution is not always easy or even possible in a closed form. Also, the questions of existence of a solution, and of multiple solutions, arise. Solving this problem is a least requirement for most robot control systems.

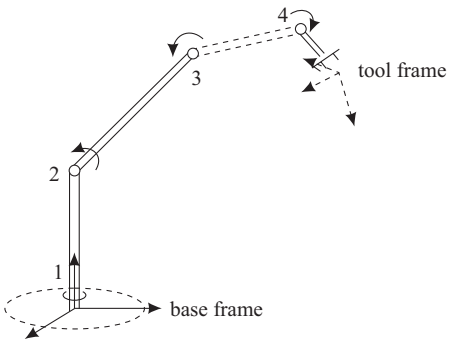

Fig. 16.1 An exemplar robot manipulator.

16.2.3 Dynamics

Dynamics is a field of study devoted to studying the forces required to cause motion. In order to accelerate a manipulator from rest, glide at a constant end-effector velocity, and finally decelerate to a stop, a complex set of torque functions must be applied by the joint actuators. In dynamics not only the geometrical properties (kinematics) are used, but also the physical properties of the robot are taken into account. Take for instance the weight (inertia) of the robot arm, which determines the force required to change the motion of the arm. The dynamics introduces two extra problems to the kinematic problems.

- The robot arm has a 'memory'. Its responds to a control signal depends also on its history (e.g. previous positions, speed, acceleration).
- If a robot grabs an object then the dynamics change but the kinematics don't. This is because the weight of the object has to be added to the weight of the arm (that's why robot arms are so heavy, making the relative weight change very small).

16.2.4 Trajectory Generation

To move a manipulator from here to there in a smooth, controlled fashion each joint must be moved via a smooth function of time. Exactly how to compute these motion functions is the problem of trajectory generation.

In the first section of this chapter we will discuss the problems associated with the positioning of the end-effector (in effect, representing the inverse kinematics in combination with sensory transformation).

16.2.5 End-Effector Positioning

The final goal in robot manipulator control is often the positioning of the hand or end-effector in order to be able to, e.g., pick up an object. With the accurate robot arm that are manufactured, this task is often relatively simple, involving the following steps:

- Determine the target coordinates relative to the base of the robot. Typically, when this position is not always the same, this is done with a number of fixed cameras or other sensors which observe the work scene, from the image frame determine the position of the object in that frame, and perform a pre-determined coordinate transformation;

- With a precise model of the robot (supplied by the manufacturer), calculate the joint angles to reach the target (i.e., the inverse kinematics). This is a relatively simple problem;
- Move the arm (dynamics control) and close the gripper. Gripper control is not a trivial matter at all, but we will not focus on that.

16.2.5a Involvement of Neural Networks

So if these parts are relatively simple to solve with a high accuracy, why involve neural networks? The reason is the applicability of robots. When 'traditional' methods are used to control a robot arm, accurate models of the sensors and manipulators (in some cases with unknown parameters which have to be estimated from the system's behavior; yet still with accurate models as starting point) are required and the system must be calibrated. Also, systems, which suffer from wear-and-tear, need frequent recalibration or parameter determination. Finally, the development of more complex (adaptive!) control methods allows the design and use of more flexible (i.e., less rigid) robot systems, both on the sensory and motor side.

16.2.6 Camera-Robot Coordination in Function Approximation

The system we focus on in this section is a work floor observed by fixed cameras, and a robot arm. The visual system must identify the target as well as determine the visual position of the end-effector.
The target position x^{target} together with the visual position of the hand x^{hand} are input to the neural controller $N(\bullet)$. This controller then generates a joint position θ for the robot:

$$\theta = N(x^{\text{target}}, x^{\text{hand}}) \qquad ...(16.1)$$

We can compare the neurally generated θq with the optimal θq_0 generated by a fictitious perfect controller $R(\bullet)$:

$$\theta_0 = R(x^{\text{target}}, x^{\text{hand}}) \qquad ...(16.2)$$

The task of learning is to make the N generate an output 'close enough' to θ_0.

There are two problems associated with teaching $N(\bullet)$:

1. Generating learning samples which are in accordance with eq. (8.2). This is not trivial, since in useful applications $R(\bullet)$ is an unknown function. Instead, a form of self-supervised or unsupervised learning is required. Some examples to solve this problem are given below;
2. Constructing the mapping $N(\bullet)$ from the available learning samples. When the (usually randomly drawn) learning samples are available, a neural network uses these samples to represent the whole input space over which the robot is active. This is evidently a form of interpolation, but has the problem that the input space is of a high dimensionality, and the samples are randomly distributed.

We will discuss three fundamentally different approaches to neural networks for robot end-effector positioning. In each of these approaches, a solution will be found for both the learning sample generation and the function representation.

16.2.6a Approach-1: Feed-forward Networks

When using a feed-forward system for controlling the manipulator, a self-supervised learning system must be used. One such a system has been reported by Psaltis, Sideris and Yamamura (1988). Here, the network, which is constrained to two-dimensional positioning of the robot arm, learns by experimentation. Three methods are proposed:

1. **Indirect learning:** In indirect learning, a Cartesian target point x in world coordinates is generated, e.g., by a two cameras looking at an object. This target point is fed into the network, which generates an angle vector θ. The manipulator moves to position θ, and the cameras determine the new position x' of the end-effector in world coordinates. This x' again is input to the network, resulting in θ'. The network is then trained on the error $\varepsilon_1 = \theta - \theta'$ (see Fig. 16.2).

   However, minimization of ε_1 does not guarantee minimization of the overall error $\varepsilon = x - x'$. For example, the network often settles at a 'solution' that maps all x's to a single θ (i.e., the mapping II).
2. **General learning:** The method is basically very much like supervised learning, but here the plant input θ must be provided by the user. Thus the network can directly minimize $|\theta - \theta'|$. The success of this method depends on the interpolation capabilities of the network. Correct choice of θ may pose a problem.

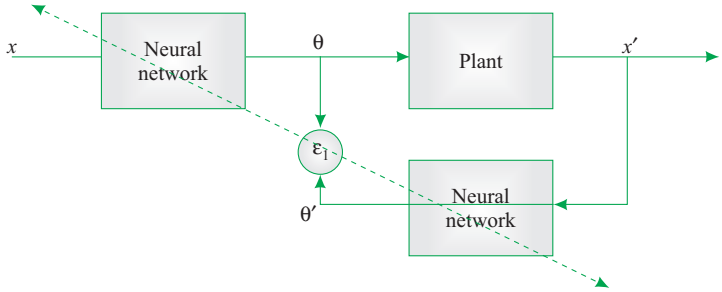

Fig. 16.2 Indirect learning system for robotics. In each cycle, the network is used in two different places: first in the forward step, then for feeding back the error.

3. **Specialized learning:** Keep in mind that the goal of the training of the network is to minimize the error at the output of the plant: $\in = x - x'$. We can also train the network by 'backpropagating' this error trough the plant (compare this with the backpropagation of the error in Chapter 12). This method requires knowledge of the Jacobian matrix of the plant. A Jacobian matrix of a multidimensional function F is a matrix of partial derivatives of F, i.e., the multidimensional form of the derivative. For example, if we have $Y = F(x)$, i.e.,

$$y_1 = f_1(x_1, x_2, ..., x_n)$$
$$y_2 = f_2(x_1, x_2, ..., x_n)$$
$$\vdots$$
$$y_m = f_m(x_1, x_2, ..., x_n)$$

then

$$\delta y_1 = \frac{\partial f_1}{\partial x_1}\delta x_1 + \frac{\partial f_1}{\partial x_2}\delta x_2 + \ldots + \frac{\partial f_1}{\partial x_n}\delta x_n$$

$$\delta y_2 = \frac{\partial f_2}{\partial x_1}\delta x_1 + \frac{\partial f_2}{\partial x_2}\delta x_2 + \ldots + \frac{\partial f_2}{\partial x_n}\delta x_n$$

$$\vdots$$

$$\delta y_m = \frac{\partial f_m}{\partial x_1}\delta x_1 \frac{\partial f_m}{\partial x_2}\delta x_2 + \ldots + \frac{\partial f_m}{\partial x_n}\delta x_n$$

or

$$\delta Y = \frac{\partial F}{\partial X}\delta X \qquad \ldots (16.3)$$

Eq. (16.3) is also written as

$$\delta Y = J(X)\ \delta X \qquad \ldots (16.4)$$

where J is the Jacobian matrix of F. so, the Jacobian matrix can be used to calculate the change in the function when its parameters change.

Now, in this case we have

$$J_{ij} = \left[\frac{\partial P_i}{\partial \theta_i}\right] \qquad \ldots(16.5)$$

where $P_i(\theta)$ the ith element of the plant output for input θ. The learning rule applied here regards the plant as an additional and unmodifiable layer in the neural network. The total error $\in = x - x'$is propagated back through the plant by calculating the δ_j:

$$\delta_j = F(s_j)\sum_i \delta_i \frac{\partial P_i(\theta)}{\partial \theta_j} \qquad \ldots(16.6)$$

$$\delta_i = x - x'$$

where I is used to change the scalar θ_j into a vector. When the plant is an unknown function, $\frac{\partial P_i(\theta)}{\partial \theta_j}$ can be approximated by

$$\frac{\partial P_i(\theta)}{\partial \theta_j} \approx \frac{P_i(\theta + h\theta_j e_j) - P_i(\theta)}{\partial \theta_j} \qquad \ldots(16.7)$$

This approximate derivative can be measured by slightly changing the input the plant and measuring the changes in the output.

A somewhat similar approach is taken in (Krose, Korst, and Groen, 1990) and (Smagt and Krose, 1991). Again a two-layer feed-forward network is trained with back-propagation. However, instead of

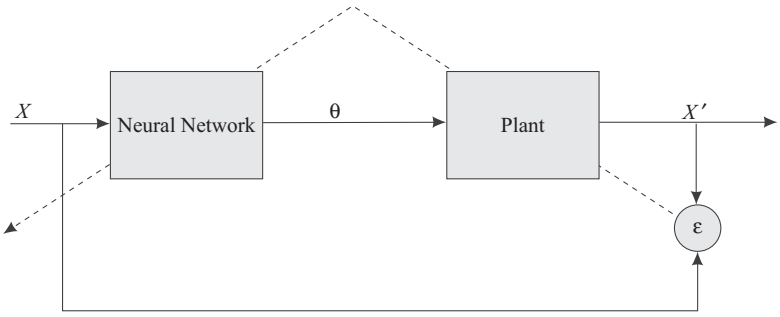

Fig. 16.3 The system used for specialized learning.

calculating a desired output vector the input vector which should have invoked the current output vector is reconstructed, and back-propagation is applied to this new input vector and the existing output vector.

The configuration used consists of a monocular manipulator, which has to grasp objects. Due to the fact that the camera is situated in the hand of the robot, the task is to move the hand such that the object is in the centre of the image and has some predetermined size (in a later article, a biologically inspired system is proposed (Smagt, Krose, and Groen, 1992) in which the visual flow-field is used to account for the monocularity of the system, such that the dimensions of the object need not to be known anymore to the system).

One step towards the target consists of the following operations:

1. Measure the distance from the current position to the target position in camera domain, *x;*
2. Use this distance, together with the current state θ of the robot, as input for the neural network. The network then generates a joint displacement vector Δθ;
3. Send Δθ to the manipulator;
4. Again measure the distance from the current position to the target position in camera domain, x';
5. Calculate the move made by the manipulator in visual domain, $x - R_i^{l+1} x'$, where R_i^{l+1} is the rotation matrix of the second camera image with respect to the first camera image;
6. Teach the learning pair $(x - R_i^{l+1} x', \theta; \Delta\theta)$ to the network.

This system has shown to learn correct behavior in only tens of iterations, and to be very adaptive to changes in the sensor or manipulator (Smagt & Krose, 1991; Smagt, Groen, & Krose, 1993).

By using a feed-forward network, the available learning samples are approximated by a single, smooth function consisting of a summation of sigmoid functions. A feed-forward network with one layer of sigmoid units is capable of representing practically any function. But how are the optimal weights determined in finite time to obtain this optimal representation?

Experiments have shown that, although a reasonable representation can be obtained in a short period of time, an accurate representation of the function that governs the learning samples is often not feasible or extremely difficult (Jansen et al., 1994). The reason for this is the global character of the approximation obtained with a feed-forward network with sigmoid units: every weight in the network has a global effect on the final approximation that is obtained.

Building local representations is the obvious way out: every part of the network is responsible for a small subspace of the total input space. Thus accuracy is obtained locally (keep it small and simple). This is typically obtained with a Kohonen network.

16.2.6b Approach 2: Topology Conserving Maps

Ritter, Martinetz, and Schulten (1989) describe the use of a Kohonen-like network for robot control. We will only describe the kinematics part, since it is the most interesting and straightforward.

The system described by Ritter et al. consists of a robot manipulator with three degrees of freedom (orientation of the end-effector is not included), which has to grab objects in 3D-space. The system is observed by two fixed cameras which output their (x, y) coordinates of the object and the end effector (see Fig. 16.4).

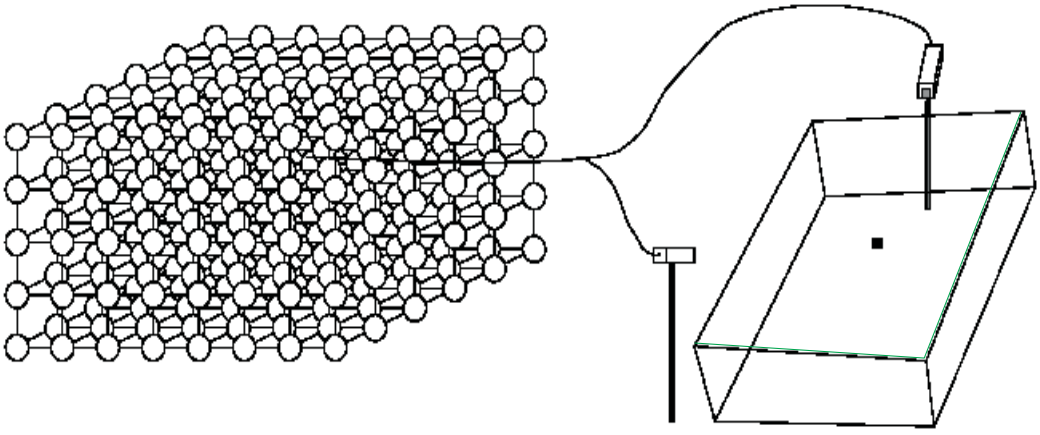

Fig. 16.4 A Kohonen network merging the output of two cameras.

Each run consists of two movements. In the gross move, the observed location of the object x (a four-component vector) is input to the network. As with the Kohonen network, the neuron k with highest activation value is selected as winner, because its weight vector w_k is nearest to x.

The neurons, which are arranged in a 3-dimensional lattice, correspond in a 1–1 fashion with subregions of the 3 D workspace of the robot, i.e., the neuronal lattice is a discrete representation of the workspace. With each neuron a vector θ and Jacobian matrix A are associated. During gross move θ_k is fed to the robot which makes its move, resulting in retinal coordinates x_g of the end-effector.

To correct for the discretization of the working space, an additional move is made which is dependent of the distance between the neuron and the object in space $w_k - x$; this small displacement in Cartesian space is translated to an angle change using the Jacobian A_k:

$$\theta^{\text{final}} = \theta_k + A_k(x - w_k) \qquad ...(16.8)$$

which is a first-order Taylor expansion of θ^{final}. The final retinal coordinates of the end-effector after this fine move are in x_f.

Learning proceeds as follows: when an improved estimate $(\theta, A)^*$ has been found, the following adaptations are made for all neurons j:

$$w_j^{\text{new}} = w_j^{\text{old}} + \gamma(t)\, g_{jk}(t)\, (x - w_j^{\text{old}})$$

$$(\theta, A)_j^{\text{new}} = (\theta, A)_j^{\text{old}} + \gamma'(t)\, g'_{jk}(t)\, ((\theta, A)_j^* - (\theta, A)_j^{\text{old}})$$

If $g_{jk}(t) = g'_{jk}(t) = \delta_{jk}$, this is similar to perceptron learning. Here, as with the Kohonen learning rule, a distance function is used such that $g_{jk}(t)$ and $g'_{jk}(t)$ are Gaussians depending on the distance between neurons j and k with a maximum at $j = k$. An improved estimate $(\theta, A)^*$ is obtained as follows:

$$\theta^* = \theta_k + A_k(x - x_f) \qquad ...(16.9)$$

$$A^* = A_k + A_k(x - w_k - x_f + x_g)\frac{(x_f - x_g)^T}{\|x_f - x_g\|^2} = A_k + (\Delta\theta - A_k\Delta x)\frac{\Delta x^T}{\|\Delta x\|^2} \qquad ...(16.10)$$

In eq. (16.9), the final error $x - x_f$ in Cartesian space is translated to an error in joint space via multiplication by A_k. This error is then added to θ_k to constitute the improved estimate θ^* (steepest descent minimization of error).

In eq. (16.10), $\Delta x = x_f - f_g$, i.e., the change in retinal coordinates of the end-effector due to the fine movement, and $\Delta\theta = A_k(x - w_k)$, i.e., the related joint angles during fine movement. Thus eq. (16.10) can be recognized as an error-correction rule of the Widrow-Hoff type for Jacobians A. It appears that after 6,000 iterations the system approaches correct behavior, and that after 30,000 learning steps no noteworthy deviation is present.

16.2.7 Robot Arm Dynamics

While end-effector positioning via sensor{robot coordination is an important problem to solve, the robot itself will not move without dynamic control of its limbs. Again, accurate control with non-adaptive controllers is possible only when accurate models of the robot are available, and the robot is not too susceptible to wear-and-tear. This requirement has led to the current-day robots that are used in many factories. But the application of neural networks in this field changes these requirements.

One of the first neural networks which succeeded in doing dynamic control of a robot arm was presented by Kawato, Furukawa, and Suzuki (1987). They describe a neural network which generates motor commands from a desired trajectory in joint angles. Their system does not include the trajectory generation or the transformation of visual coordinates to body coordinates.

The network is extremely simple. In fact, the system is a feed-forward network, but by carefully choosing the basis functions, the network can be restricted to one learning layer such that finding the optimal is a trivial task. In this case, the basis functions are thus chosen that the function that is approximated is a linear combination of those basis functions.

Dynamics model. The manipulator used consists of three joints as the manipulator in Fig. 16.1 without wrist joint. The desired trajectory $\theta_d(t)$, which is generated by another subsystem, is fed into the inverse-dynamics model (Fig. 16.5). The error between $\theta_d(t)$ and $\theta(t)$ is fed into the neural model.

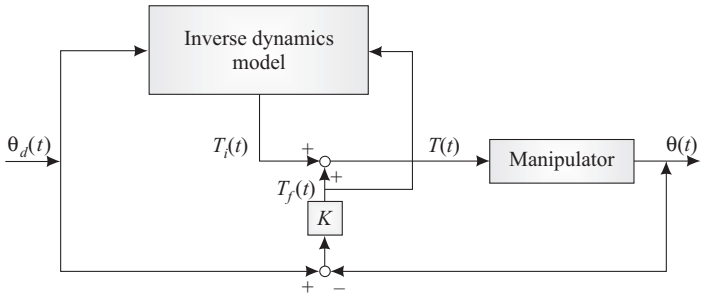

Fig. 16.5 The neural model proposed by Kawato et al.

The neural model, which is shown in Fig. 16.6, consists of three perceptrons, each one feeding in one joint of the manipulator. The desired trajectory $\theta_d = (\theta_{d1}, \theta_{d2}, \theta_{d3})$ is feed into 13 nonlinear subsystems. The resulting signals are weighted and summed, such that

$$T_{ik}(t) = \sum_{i=1}^{13} w_{lk} x_{lk} \quad (k = 1, 2, 3) \qquad ...(16.11)$$

with

$$x_{l1} = f_1(\theta_{d1}(t), \theta_{d2}(t), \theta_{d3}(t))$$

$$x_{l2} = x_{l3} = g_l(\theta_{d1}(t), \theta_{d2}(t), \theta_{d3}(t))$$

and f_l and g_l as in Table 16.1.

The feedback torque $T_f(t)$ in Fig. 16.5 consists of

$$T_{fk}(t) = K_{pk}(\theta_{dk}(t) = \theta_k(t)) + K_{vk} \frac{d\theta(t)}{dt}, \quad (k\text{–}1,2,3)$$

$$K_{vk} = 0 \text{ unless } |\theta_k(t) - \theta_{dk}(\text{objective point})| < \varepsilon$$

The feedback gains K_p and K_v were computed as $(517.2, 746.0, 191.4)^T$ and $(16.2, 37.2, 8.4)^T$.

Next, the weights adapt using the delta rule

$$\gamma \frac{dw_{ik}}{dt} = x_{ik} T_1 = x_{ik}(T_{jk} - T_{ik}), \quad (k = 1,2,3) \qquad ...(16.12)$$

A desired move pattern is shown in Fig. 16.7. After 20 minutes of learning the feedback torques are nearly zero such that the system has successfully learned the transformation. Although the applied patterns are very dedicated, training with a repetitive pattern sin $(w_k t)$, with $w_1 : w_2 : w_3 = 1 : \sqrt{2} : \sqrt{3}$ is also successful.

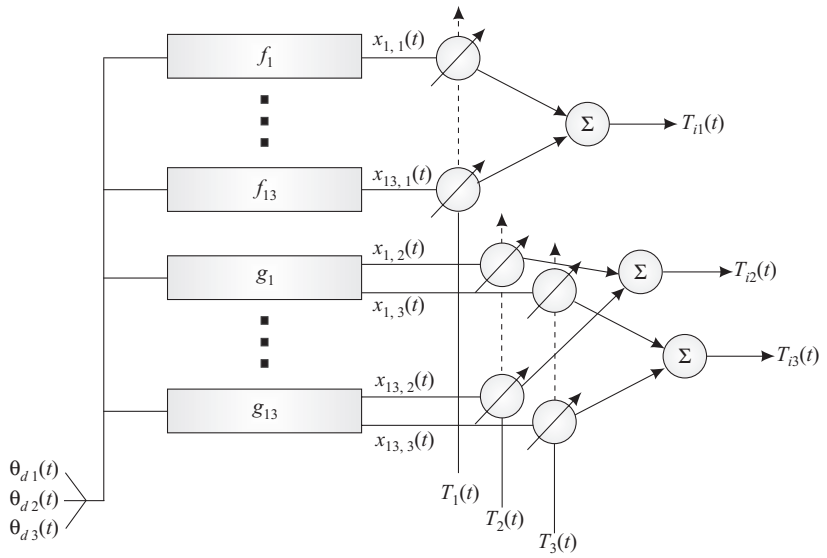

Fig. 16.6 The neural network used by Kawato et al. There are three neurons, one per joint in the robot arm. Each neuron feeds from thirteen nonlinear subsystems. The upper neuron is connected to the rotary base joint (cf. joint 1 in Figure 16.1), the other two neurons to joints 2 and 3.

Table 16.1: Nonlinear transformations used in the Kawato model.

l	$f_1(\theta_1, \theta_2, \theta_3)$	$g_1(\theta_1, \theta_2, \theta_3)$
1	$\ddot{\theta}_1$	$\ddot{\theta}_2$
2	$\ddot{\theta}_1 \sin^2 \theta_2$	$\ddot{\theta}_3$
3	$\ddot{\theta}_1 \cos^2 \theta_2$	$\ddot{\theta}_2 \cos \theta_3$
4	$\ddot{\theta}_1 \sin^2 (\theta_2 + \theta_3)$	$\ddot{\theta}_3 \cos \theta_3$
5	$\ddot{\theta}_1 \cos^2 (\theta_2 + \theta_3)$	$\dot{\theta}_1^2 \sin \theta_2 \cos \theta_2$
6	$\ddot{\theta}_1 \sin \theta_2 \sin (\theta_2 + \theta_3)$	$\dot{\theta}_1^2 \sin (\theta_2 + \theta_3) \cos (\theta_2 + \theta_3)$
7	$\ddot{\theta}_1 \dot{\theta}_2 \sin \theta_2 \cos \theta_2$	$\dot{\theta}_1^2 \sin \theta_2 \cos (\theta_2 + \theta_3)$
8	$\ddot{\theta}_1 \dot{\theta}_2 \sin (\theta_2 + \theta_3) \cos (\theta_2 + \theta_3)$	$\dot{\theta}_1^2 \cos \theta_2 \sin (\theta_2 + \theta_3)$
9	$\ddot{\theta}_1 \dot{\theta}_2 \sin \theta_2 \cos (\theta_2 + \theta_3)$	$\dot{\theta}_2^2 \sin \theta_3$
10	$\ddot{\theta}_1 \dot{\theta}_2 \cos \theta_2 \sin (\theta_2 + \theta_3)$	$\dot{\theta}_3^2 \sin \theta_3$
11	$\ddot{\theta}_1 \dot{\theta}_3 \sin (\theta_2 + \theta_3) \cos (\theta_2 + \theta_3)$	$\dot{\theta}_2 \ddot{\theta}_3 \sin \theta_3$
12	$\ddot{\theta}_1 \dot{\theta}_3 \sin \theta_2 \cos (\theta_2 + \theta_3)$	$\dot{\theta}_2$
13	$\ddot{\theta}_1$	$\dot{\theta}_3$

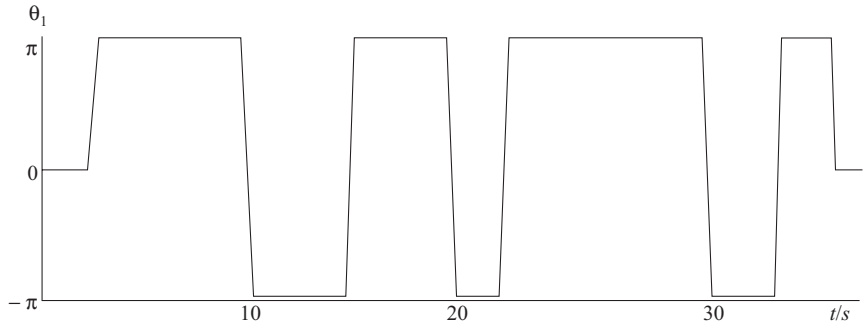

Fig. 16.7 The desired joint pattern for joint 1. Joints 2 and 3 have similar time patterns.

The usefulness of neural algorithms is demonstrated by the fact that novel robot architectures, which no longer need a very rigid structure to simplify the controller, are now constructed.

16.3 DETECTION OF TOOL BREAKAGE IN MILLING OPERATIONS

The recent trend in manufacturing is to achieve integrated and self-adjusting machining systems, which are capable of machining varying parts without the supervision of operators. The absence of human supervision requires on-line monitoring of machining operation, ensuring safe and efficient metal removal rate and taking corrective actions in the event of failures and disturbances (Yusuf, 1998). One of the most important monitoring requirements is a system capable of detecting tool breakages on-line. Unless recognized in time, tool breakage can lead to irreparable damage to the workpiece and possibly to the machine tool itself.

The cutting force variation characteristics of normal and broken tools are different. With the normal and broken tool cutting force variation signals is possible to train neural networks. The milling operations can be monitored with the neural network, after training. The use of adaptive resonance theory (ART) type neural network was evaluated for detections of tool breakage, in this study (Ibrahim and Mclaughlin, 1993). Also simulation-based training is proposed to reduce the cost of preparing the systems that monitor the real cutting signals.

Neural networks with parallel processing capability and robust performances provide a new approach to adaptive pattern recognition. Adaptive Resonance Theory (ART 2) architectures are neural networks that carry out stable self-organization of recognition codes for arbitrary sequence of input patterns. Artificial neural networks refer to a group of architectures of the brain (Cheng and Sheng, 1995). Neural networks are also classified as supervised and unsupervised according to their learning characteristics. In unsupervised learning, the neural network classifies the signals by itself.

In this section, ART type unsupervised neural network paradigm was used for detection of tool breakage. ART paradigm was used for the following reasons.

(a) The training of paradigm is much faster than the back-propagation technique;

(b) The back-propagation technique generalizes the given information in order to store it inside the initially selected hidden layers. The back propagation technique cannot give reliable decisions on the sufficiency of previous training; and

(c) ART has very important advantage since it can be trained in the field and continuously updates previous experience.

The unsupervised ART neural networks can monitor the signal based on previous experience and can update itself automatically while it is monitoring the signals (Carpenter and Grossberg, 1991). When as ART network receives an input pattern, this bottom-up pattern is compared to the top-down, or already known patterns. If the input pattern is matched with known pattern in memory, the weights of the model are changed to update the category. If the new pattern cannot be classified in a known category, it is coded and classified as a new category.

Another important issue is the training of the neural network. It is extremely expensive and time consuming to collect cutting force data at different cutting conditions with normal and broken tools. To overcome this problem, simulation-based training of neural networks was introduced. Simulated data was used to select the best vigilance of the ART 2 type neural network and to evaluate the performance of paradigm. The theoretical background of ART 2 type neural network, the proposed data monitoring system and their performance is presented in the paper.

16.3.1 Unsupervised Adaptive Resonance Theory (ART) Neural Networks

The theory of adaptive resonance networks was first introduces by Carpenter and Grossberg, (1987). Adaptive resonance occurs when the input to a network and the feed back expectancies match. The ART 2 neural networks developed by Carpenter and Grossberg (1991) self-organize recognition codes in real time.

The basic ART 2 architectures consist of two types of nodes, the short term memory (STM) nodes, which are temporary and flexible, and the long term memory (LTM) nodes; which are permanent and stable. The input pattern (i) is received by the STM, where it is normalized, matched, learned, and stored in the LTM (z_{ji}). The STM is divided into two sets of nodes, F_1 and F_2. The STM F_1 nodes are used for normalization, control, gain and learning procedures. The F_1 field in ART 2 includes a combination of normalization and noise suppression, in addition to the comparison of the bottom-up and top-down signals needed for the reset mechanism. To accomplish this, F_1 uses the following equations to calculate the nodes:

$$u_i = \frac{v_i}{e + \|v\|} \qquad ...(16.13)$$

$$w_i = S_i + au_i \qquad ...(16.14)$$

$$p_i = u_i + \Sigma q(y_i) z_{ji} \qquad ...(16.15)$$

$$q_i = \frac{p_i}{e + \|p\|} \qquad ...(16.16)$$

$$v_i = f(x_i) + bf(q_i) \qquad ...(16.17)$$

$$x_i = \frac{w_i}{e + \|w\|} \qquad ...(16.18)$$

Here $\|p\|$, $\|v\|$ and $\|w\|$ denote the norms of the vectors p, v and w, and s_i is the input. The non-linear signal function in equation (5) is used for noise suppression . the activation function (f) is given by the equation.

$$f(x) = \begin{cases} x & \text{if } x \geq \theta \\ 0 & \text{if } 0 \leq x \geq \theta \end{cases} \quad ...(16.19)$$

where θ is an appropriate constant. The function f filters the noise from the signal. θ can be set to zero for the case where filtering is not desired. The constants a, b, and e are selected based on the particular application.

The STM F_2 nodes are used for the matching procedure. F_2 equations select or activate nodes in the LTM. When F_2 chooses a node, all other nodes in the LTM are inhibited, and only one is allowed to interact with the STM. The node that gives the largest sum with the F_1, F_2 input pattern (bottom-up) is the key property that is used for node selection.

Bottom-up inputs are calculated as in ART 2 [Fauselt, 2004]

$$T_j = \sum_i p_i z_{ji} \quad ...(16.20)$$

The j^{th} node is selected if equation (16.20) is satisfied.

$$T_j = \max \{ T_j : 1, 2 \ldots N\} \quad ...(16.21)$$

Competition on F_2 results in contrast enhancement where a single winning node is chosen each time. The output function of F_2 is given by

$$g(y_i) = \begin{cases} d & \text{if } T_i : j = 1, 2, \ldots N \\ 0 & \text{otherwise} \end{cases} \quad ...(16.22)$$

Equation (3) takes the following form:

$$p_i = \begin{cases} u_i & \text{if } F_2 \text{ is inactive} \\ u_i + dz_{ij} & \text{if } j\text{th node is } F_2 \text{ is active} \end{cases} \quad ...(16.23)$$

The bottom-up and top-down LTM equations are

bottom-up ($F_1 \rightarrow F_2$) :

$$\frac{dz_{ij}}{dt} = g(y_j)\,[p_i - z_{ij}] \quad ...(16.24)$$

top–bottom ($F_2 \rightarrow F_1$) :

$$\frac{dz_{ij}}{dt} = g(y_j)\,[p_i - z_{ij}] \quad ...(16.25)$$

When F_2 is active, then equations (16.24) and (16.25) are modified from equation (16.22) to:

$$\frac{dz_{ij}}{dt} = d[p_i - z_{ij}] \qquad ...(16.26)$$

where d is a constant ($0 < d < 1$). An orienting ART 2 sub system is used to decide, if a new pattern can be matched to a known pattern by comparing with a given vigilance parameter, ρ:

$$r_i = \frac{u_i + cp_i}{e + ||u|| + ||cp||} \qquad ...(16.27)$$

If $||r|| < \rho - e$, then F_2 resets another node. If $||r|| \geq \rho - e$, a match has been found and the new pattern is learned by the system. The LTM node weights are recalculated and the pattern is learned by the system. If no match has been found after all nodes have been activated, a new node is created, and the new pattern is stored.

16.3.2 Results and Discussion

The experimental data was collected with a fo1ur flute end mill of 12.07 mm diameter at various cutting conditions. The ART neural network monitored the profile of the resultant force in different tests. In the three tests, experiments were done at different feed rates with the good and broken tool. The spindle speed, feed rate, and depth of cut of these different conditions are out lined in Tables 16.2-16.5. The neural network did not have any prior information at the beginning of each test. In each one, the neural network inspected the resultant force profile and placed it into a category or initiated a new category if it was found to be different.

The vigilance of the ART 2 selected either 0.96 or 0.98 in all the tests. The ART assigned 2, 2, 3, 1 and 3 different categories for the good tool. For the broken tool 2, 1, 1, 1 and 3 different categories were selected. In all the tests, the neural network classified the good and broken tools in different categories. As seen in Tables 16.2-16.5, the neural network generated only one category in the 2nd (Table 16.3) , 3rd (Table16.4) and 4th (Table 16.5) tests for the broken tools. On the other hand, the neural network assigned more nodes to the signal of a good tool with offset. It indicates that the broken tool signals are more similar to each other at different cutting conditions compared to the force patterns of normal tools.

Table-16.2 Classification of experimental data with the ART. Vigilance of the neural network was 0.96. The ART used four categories to classify all of the data.

Spindle speed (rpm)	*Depth of cut (mm)*	*Feed rate mm/min*	*Tool condition*	*Category*
500	1.016	50.8	G	1
500	1.016	50.8	B	2
500	1.016	101.6	G	1
500	1.016	101.6	B	3
500	1.016	203.2	G	1
500	1.016	203.2	B	3
500	1.016	254	G	4
500	1.016	254	B	5

Table16.3: Classification of experimental data with the ART. Vigilance of the neural network was 0.96. The ART used three categories to classify all of the data.

Spindle speed (rpm)	*Depth of cut (mm)*	*Feed rate mm/min*	*Tool condition*	*Category*
500	1.524	50.8	G	1
500	1.524	50.8	B	2
500	1.524	101.6	G	3
500	1.524	101.6	B	2
500	1.524	203.2	G	1
500	1.524	203.2	B	2
500	1.524	254	G	3
500	1.524	254	B	2

Table 16.4: Classification of experimental data with the ART 2. Vigilance of the neural network was 0.98. The ART 2 used four categories to classify all of the data.

Spindle speed (rpm)	*Depth of cut (mm)*	*Feed rate mm/min*	*Tool condition*	*Category*
700	1.016	50.8	G	1
700	1.016	50.8	B	2
700	1.016	101.6	G	3
700	1.016	101.6	B	2
700	1.016	203.2	G	4
700	1.016	203.2	B	2
700	1.016	254	G	4
700	1.016	254	B	2

Table 16.5: Classification of experimental data with the ART 2. Vigilance of the neural network was 0.96. The ART 2 used two categories to classify all of the data.

Spindle speed (rpm)	*Depth of cut (mm)*	*Feed rate mm/min*	*Tool condition*	*Category*
700	1.524	50.8	G	1
700	1.524	50.8	B	2
700	1.524	101.6	G	1
700	1.524	101.6	B	2
700	1.524	203.2	G	1
700	1.524	203.2	B	2
700	1.524	254	G	1
700	1.524	254	B	2

The ART gained first experience on the simulation data and later, the neural network inspected the incoming signals and continued to assign new categories when different types of signals were encountered. After simulation training, the neural network started to monitor the experimental data collected at different conditions. The studies focused on selection of the best vigilance, which requires a minimum number of nodes and has acceptable error rate. When the vigilance of 0.98 is used, the network classified the perfect too input data into seven different categories and classified the broken tool input data into four different categories.

QUESTION BANK

1. Enumerate various applications of neural networks.
2. Explain the application self-supervised learning to control manipulator.
3. Explain the application of Kohonon network for robot control.
4. Explain the application of neural network for robot arm dynamics.
5. Explain the application of ART for machining applications.

REFERENCES

1. J.J. Craig, *Introduction to Robotics*, Addison-Wesley Publishing Company, 1989.
2. D. Psaltis, A. Sideris, and A.A. Yamamura, A multilayer neural network controller, *IEEE Control Systems Magazine*, Vol. 8, No. 2, pp. 17-21, 1988.
3. B.J.A. Krose, M.J. Korst, and F.C.A. Groen, Learning strategies for a vision based neural controller for a robot arm, In O. Kaynak (Ed.), IEEE International Workshop on Intelligent Motor Control, pp. 199-203, IEEE, 1990.
4. P.P. Smagt, and B.J.A. Krose, A real-time learning neural robot controller, In T. Kohonen, K. Makisara, O. Simula, & J. Kangas (Eds.), Proceedings of the 1991 International Conference on Artificial Neural Networks, North-Holland/Elsevier Science Publishers, pp. 351-356, 1991.
5. B.J.A. Krose, M.J. Korst, and F.C.A. Groen, Using time-to-contact to guide a robot manipulator, In Proceedings of the 1992 IEEE/RSJ International Conference on Intelligent Robots and Systems, 1992.
6. P.P. Smagt, and B.J.A. Krose, A real-time learning neural robot controller, In T. Kohonen, K.M. akisara, O. Simula, & J. Kangas (Eds.), Proceedings of the 1991 International Conference on Artificial Neural Networks, North Holland/Elsevier Science Publishers, pp. 351-356,1991.
7. B.J.A. Krose, F.C.A. Groen, and B.J.A. Krose, Robot Hand-Eye Coordination Using Neural Networks, Tech. Rep. Nos. CS-93-10, Department of Computer Systems, University of Amsterdam, 1993.
8. A. Jansen, P.P. Smagt, and F.C.A. Groen, Nested networks for robot control, In A.F. Murray (Ed.), Neural Network Applications, Kluwer Academic Publishers, 1994.
9. H.J. Ritter, T.M. Martinetz, and K.J. Schulten, Topology-conserving maps for learning visuo-motor-coordination, *Neural Networks*, Vol. 2, pp. 159-168, 1989.
10. Kawato, M., Furukawa, K., & Suzuki, R. (1987). A hierarchical neural-network model for control and learning of voluntary movement. Biological Cybernetics, 57, 169-185.
11. A. Yusuf "In-processs detection of tool breakages using time series monitoring of cutting forces", *Int. J. Mach.Tools Manufact*, Vol. 28, No. 2, pp. 157-172, 1998.
12. N.T. Ibrahim and C. Mclaughlin, "Detection of tool breakage in milling – II, The neural network approach. *Int. J. Mach.Tools Manufact*, Vol. 33, No. 4, pp. 545-558, 1993.
13. A.H.Cheng and F. Sheng, "Adaptive hamming net. A Fast learning ART 1 model without searching", *Int. J. of Neural Networks*, Vol. 8, No. 4, pp. 605-618, 1995.

14. Carpenter and Grossberg, "ART 2-A: Adaptive resonance Algorithm for rapid category learning and recognition", *Int. J. of Neural Network*, Vol. 4, pp. 493-504, 1991.
15. L. Fauselt, *Fundamentals of Neural Networks*, *Architectures*, *Algorithms*, *and Applications*, Pearson Education, 2nd Edition, 2004.
16. G.A. Carpenter, and S. Grossberg, A massively parallel architecture for a self-organizing neural pattern recognition machine, Computer Vision, *Graphics*, *and Image Processing*, Vol. 37, pp. 54-115, 1987.
17. G.A. Carpenter, and S. Grossberg, ART 2: Self-organization of stable category recognition codes for analog input patterns, *Applied Optics*, 26(23), 4919-4930, 1987.

CHAPTER 17

Hybrid Fuzzy Neural Networks

17.1 INTRODUCTION

Neural networks and fuzzy systems try to emulate the operation of human brain. Neural networks concentrate on the structure of human brain, i.e., on the "hardware" whereas fuzzy logic systems concentrate on "software". Combining neural networks and fuzzy systems in one unified framework has become popular in the last few years.

17.2 HYBRID SYSTEMS

Designation "neurofuzzy" has several different meanings. Sometimes "neurofuzzy" is associated to hybrid systems, which act on two distinct subproblems. In that case, neural network is utilized in the first subproblem (e.g., in signal processing) and fuzzy logic is utilized in the second subproblem (e.g., in reasoning task). Normally, when talking about the neurofuzzy systems, the link between these two soft computing methods is understood to be stronger. In the following light is tried to shed on the most common different interpretations.

Hybrid systems are those for which more than one technology is employed to solve the problem. The hybrid systems are classified as:

- Sequential hybrids
- Auxiliary hybrids
- Embedded hybrids

17.2.1 Sequential Hybrid Systems

Sequential hybrid systems make use of technologies in a pipeline-like structure. The output of one technology becomes the input to another technology and so on. A simple sequential hybrid system is shown in Fig. 17.1. This is one of the weakest forms of hybridization because an integrated combination of the technologies is not present.

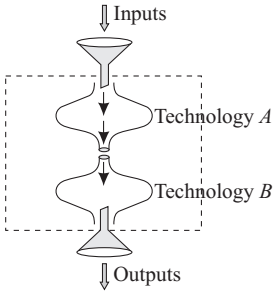

Fig. 17.1 A sequential hybrid system.

17.2.2 Auxiliary Hybrid Systems

In this, one technology calls the other as a subroutine to process or manipulate information by it. The second technology processes the information provided by the first and hands it over for further use. An auxiliary hybrid system is shown in Fig. 17.2. The auxiliary hybrid system is better than the sequential hybrid system.

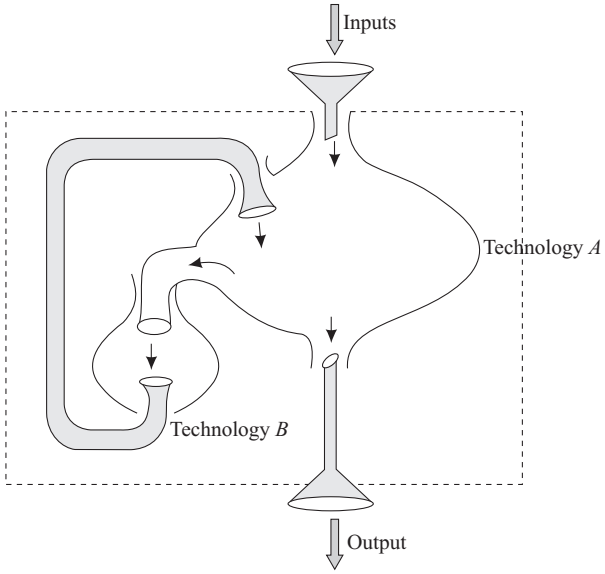

Fig. 17.2 An auxiliary hybrid system.

17.2.3 Embedded Hybrid Systems

In embedded hybrid systems, the technologies participating are integrated in such a manner that they appear intertwined. The fusion is so complete that it would appear that no technology could be used without the others for solving the problem. Fig. 17.3 illustrates an embedded hybrid system. The embedded hybrid system is better than sequential and auxiliary hybrid systems.

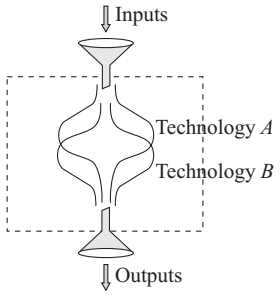

Fig. 17.3 An embedded hybrid system.

17.3 FUZZY LOGIC IN LEARNING ALGORITHMS

A common approach is to use fuzzy logic in neural networks to improve the learning ability. For example, a fuzzy control of back-propagation is illustrated in Fig. 17.3. The purpose is to achieve a faster rate of convergence by controlling the learning rate parameter with fuzzy rules. Rules are of the type:

Rule 1: *IF*(*G* o *E* is *NB*) *AND* (*C* o *G* o *E* is *NB*) *THEN* (*C* o *LP* is *NS*)

-
-
-

Rule 13: *IF*(*G* o *E* is *ZB*) *AND* (*C* o *G* o *E* is *ZE*) *THEN* (*C* o *LP* is *PS*)

-
-
-

Rule 25: *IF*(*G* o *E* is *PB*) *AND* (*C* o *G* o *E* is *PB*) *THEN* (*C* o *LP* is *NS*)

where *C* o *LP* is change of learning parameter, *G* o *E* is the gradient of the error surface, *C* o *G* o *E* is change of *G* o *E* (approximation to second order gradient), *NB*, *NS*, *ZE*, *PS* and *PB* are fuzzy sets "negative big", "negative small", "zero equal", "positive small" and "positive big". (They also incorporated in rules information about the sign change of gradient and information about the momentum constant.)

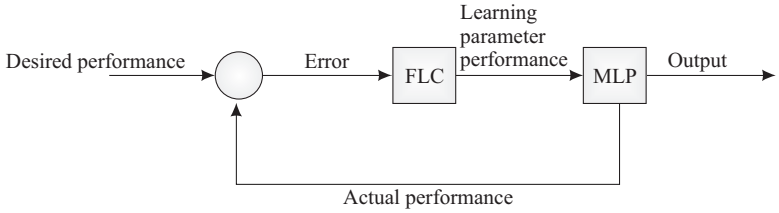

Fig. 17.4 Learning rate control by fuzzy logic. FLC is fuzzy logic controller, MLP is multiplayer perceptron.

Simulation results show that the convergence of fuzzy back propagation is faster than standard back propagation.

17.4 FUZZY NEURONS

Definition 17.1: (*hybrid neural network*) A hybrid neural network is a network with real valued inputs $x_i \in [0, 1]$, (usually membership degrees) and real valued weights $w_i \in [0, 1]$. Input and weight are combined (i.e., product is replaced) using *t*-norm

$$y = T(S(x_1, w_1), ..., S(x_d, w_d)) \quad ...(17.1)$$

$$y = S(T(x_1, w_1), ..., T(x_d, w_d)) \quad ...(17.2)$$

$T(x_i, w_i)$, *t*-conorm $S(x_i, w_i)$, or some other continuous operation. These combinations are again combined (i.e., addition is replaced) using *t*-norm, *t*-conorm, or some continuous operation. Activation function *g* can be any continuous function. If we choose linear activation function, *t*-norm (min) for addition and *t*-conorm (max) for product, we get an *AND fuzzy neuron* (17.1), and if they are chosen on the contrary, we get an *OR fuzzy neuron* (17.2). Output of (17.1) corresponds to min-max composition and (17.2) corresponds to max-min composition known from the fuzzy logic.

Another way to implement fuzzy neuron is to extend weights and/or inputs and/or outputs (or targets) to fuzzy numbers. The choices are as follows:

1. crisp input, fuzzy weights, fuzzy output (or crisp output by defuzzification).
2. fuzzy input, crisp weights, fuzzy output
3. fuzzy input, fuzzy weights, fuzzy output

which can be used to implement fuzzy IF-THEN rules. In addition there exists a type of network where the weights and targets are crisp and the inputs are fuzzy. The networks of this type are used in classification problems to map fuzzy input vectors to crisp classes.

Definition 17.2: (*regular fuzzy neural network, RFNN*) A regular fuzzy neural network is a network with fuzzy signals and/or fuzzy weights, sigmoidal activation function and all the operations are defined by extension principle.

Example 17.1: Consider a simple network $y = g(w_1x_1 + w_2x_2)$, where the inputs and weights are fuzzy numbers. We use the extension principle to calculate w_ix_i Output fuzzy set Y is computed by the Extension principle

$$Y(y) = \begin{cases} (w_1x_1 + w_2x_2)g^{-1(y)} & \text{if } 0 \le y \le 1 \\ 0 & \text{otherwise} \end{cases} \quad ...(17.3)$$

where $g^{-1}(y) = \ln y - \ln (1 - y)$ is simply the inverse function of logistic sigmoid $g(z) = 1/(1 + e^{-z})$.

The problem of regular fuzzy neural networks is that they are monotonic, which means that the fuzzy neural nets based on the extension principle are universal approximators only for monotonic functions.

Theorem 17.1: g is an increasing function of its arguments, i.e. if $x_1 \subset x_1'$ and $x_2 \subset x_2'$ (x_i, x_i', w_i are fuzzy numbers), then

$$g(w_1x_1 + w_2w_2) \subset g(w_1x_1' + w_2x_2') \quad ...(17.4)$$

Proof: Because min and max are increasing functions, then

$$(w_1x_1 + w_2x_2) \subset (w_1x_1' + w_2x_2') \quad ...(17.5)$$

which means that regular fuzzy neural network is not a universal approximator. This is a serious drawback for the networks of this type.

Definition 17.3: (*hybrid fuzzy neural network, HFNN*) A hybrid fuzzy neural network is a network with fuzzy valued inputs $x_i \in [0, 1]$, and/or fuzzy valued weights $w_i \in [0, 1]$. Input and weight are combined using t-norm $T(x_i, w_i)$, t-conorm $S(x_i, w_i)$, or some other continuous operation. These combinations are again combined using t-norm, t-conorm, or some continuous operation. Activation function g can be any continuous function.

Many researchers working with fuzzy neurons follow the basic principles described above, but there is no standard path to follow.

17.5 NEURAL NETWORKS AS PRE-PROCESSORS OR POST-PROCESSORS

One of the biggest problems with fuzzy systems is the curse of dimensionality. When the dimension of the problem increases the size of the fuzzy model (and the size of training set needed) grows exponentially. The use of more than 4 inputs may be impractical. The number of combinations of input terms (possibly also the number of rules) is

$$\prod_{i}^{d} m_i \quad ...(17.6)$$

where m_i is the number of fuzzy sets on axis i. For example, if we have five inputs and each input space is partitioned into seven fuzzy sets, the number of combinations is 16,807. Therefore, there is a strong need for data reduction. The smallest number of input variables should be used to explain a maximal amount of information.

The most common method to decrease the dimension of input space is the principal component analysis (PCA). The main goal of identifying principal components is to preserve as much relevant information as possible. Selecting M attributes from d is equivalent to selecting M basis vectors which span the new subspace, and projecting the data onto this M-dimensional subspace. Therefore, identifying principal reduce the dimensionality of a data in which there are large number of correlated variables and at the same time retaining as much as possible of the variation present in the data. This reduction is achieved via a set of linear transformations, which transform input variables to a new set of variables (uncorrelated principal components). The algorithm goes as follows:

1. compute the mean of inputs in data and subtract it off
2. calculate covariance matrix and its eigenvectors and eigenvalues

3. retain eigenvectors corresponding to the M largest eigenvalues
4. project input vectors onto the eigenvectors

Neural networks may be used to perform the dimensionality reduction. A two-layer perceptron with linear output units (number of hidden units is M, with $M < d$) which is trained to map input vectors onto themselves by minimization of sum-of-squares error is able to perform a linear principal component analysis. If two additional nonlinear hidden layers are allowed to put into the network, the network can be made to perform a non-linear principal component analysis.

17.6 NEURAL NETWORKS AS TUNERS OF FUZZY LOGIC SYSTEMS

The similarities between neural networks and fuzzy logic systems were noticed, which led to the development of neurofuzzy systems. The original purpose of neurofuzzy systems was to incorporate learning (and classification) capability to fuzzy systems or alternatively to achieve similar transparency (intelligibility) in neural networks as in fuzzy systems.

Learning is assumed to reduce design costs, increase flexibility, improve performance and decrease human intervention. If prior knowledge is unavailable and/or the plant is time-varying then the most sensible (possibly the only) solution is to utilize learning capabilities.

In the 1980s, a computationally efficient training algorithm for multi-layer neural networks was discovered. It was named *error back-propagation*. The principle of the method is that it finds the derivatives of an error function with respect to the weights in the network. The error function can then be minimized by using gradient based optimization algorithms. Since back-propagation can be applied to any feed forward network, some researchers began to represent fuzzy logic systems as feed forward networks. The idea was to use the training algorithm to adjust weights, centers and widths of membership functions.

In Fig. 17.5, the fuzzy logic system with d inputs one output approximator, is represented as a network.

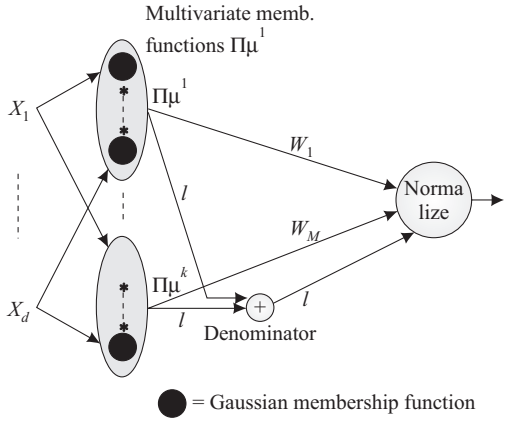

Fig. 17.5 Neurofuzzy network for back-propagation.

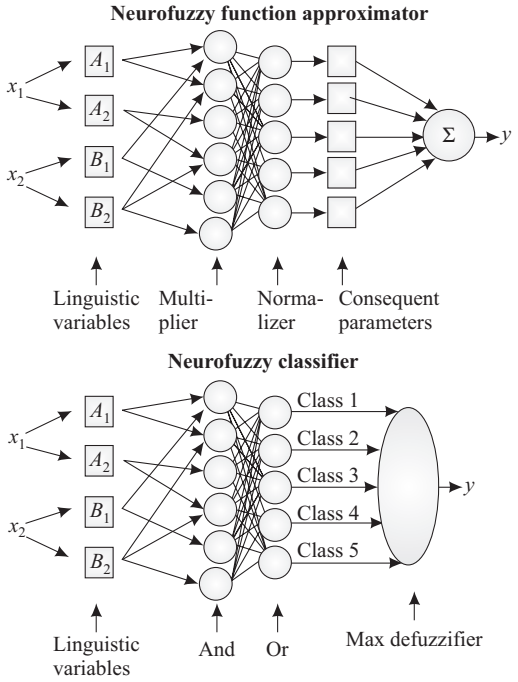

Fig. 17.6 Neurofuzzy network for function approximation and classification problems.

The most common way to represent neurofuzzy architecture is shown in Figure 17.6. Although it looks different from the network in Fig. 17.5, it is basically the same network. Only the way to illustrate network differs.

17.7 ADVANTAGES AND DRAWBACKS OF NEUROFUZZY SYSTEMS

The advantages are as follows:

1. Weights are the centers of THEN part fuzzy sets (clear meaning)
2. Other parameters (width and position of membership functions) have clear meaning
3. Initial weights can be chosen appropriately (linguistic rules)

The drawback is the curse of dimensionality.

17.8 COMMITTEE OF NETWORKS

The method of combining networks to form a *committee* has been used to improve the generalization ability of the networks. The performance of committee can be better than the performance of isolated networks. It can consist of networks with different architectures, e.g., different types of neural networks, fuzzy logic systems and conventional models.

The committee prediction (output) is calculated as an average of the outputs of the q networks:

$$y_{COM}(x) = \frac{1}{q}\sum_{i=1}^{q} y_i(x) \qquad ...(17.7)$$

The reduction of error arises from the reduced variance due to averaging.

Kosko (1991) has proposed the use of weighted average to combine different fuzzy systems that try to predict the same input-output relation. He does not restrict the form of fuzzy system to be additive or SAM system. The only difference with (17.7) is that Kosko weights fuzzy system outputs $y_i(x)$ by credibilities $w_i \in [0, 1]$, such that at least one system has nonzero credibility.

17.9 FNN ARCHITECTURE BASED ON BACK PROPAGATION

The input-output relation is numerically calculated by interval arithmetic via level sets (i.e., α-cuts) of fuzzy weights and fuzzy inputs. Next we define the strong L-R type fuzzy number, and show its good properties in interval arithmetic. While defining a cost function for level sets of fuzzy outputs and fuzzy targets, we propose a learning algorithm from the cost function for adjusting three parameters of each strong L-R type fuzzy weight. Lastly we examine the ability of proposed fuzzy neural network implementing on fuzzy if-then rules.

In the fuzzy neural networks based on BP, neurons are organized into a number of layers and the signals flow in one direction. There are no interactions and feedback loops among the neurons of same layer, Fig. 17.7 shows this model fuzzy neural network.

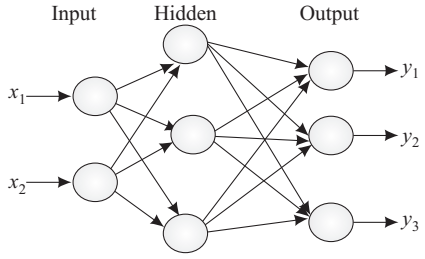

Fig. 17.7 A three layered fuzzy neural network

According to the type of inputs and weights we define three different kinds of fuzzy neural networks as follows:

I. crisp weight and fuzzy inputs;
II. fuzzy weight and crisp inputs; and
III. fuzzy weight and fuzzy inputs.

Type (III) of fuzzy feed forward neural networks is presented here. In this model, the connections between the layers may be illustrated as a matrix of fuzzy weights w_{ji}, which provides a fuzzy weight of a connection between ith neuron of the input layer, and jth neuron of the hidden layer. The total fuzzy input of jth neuron in the second layer is defined as:

$$Net_{pj} = \sum_{i=1}^{N} w_{ji} . O_{pj+} \Theta_j \quad ...(17.8)$$

Where, Net_{pj} is the total fuzzy input of the jth neuron of hidden layer, $O_{pi} = x_{pi}$ is the ith fuzzy input of that neuron, and Θ_j is fuzzy bias of the jth neuron. The fuzzy output of the jth neuron is defined with the transfer function

$$O_{pj} = f(Ney_{pj}), \quad j = 1, 2, ..., N_H \quad ...(17.9)$$

Furthermore, the fuzzy output of the kth neuron of output layer is defined as follows:

$$Net_{pk} = \sum_{j=1}^{N_H} w_{kj} . O_{pj+} \Theta_k \quad ...(17.10)$$

$$O_{pk} = f(Net_{pk}), j = 1, 2, ..., N_O \quad ...(17.11)$$

The fuzzy output is numerically calculated for level sets (i.e. α - cut) of fuzzy inputs, fuzzy weights and fuzzy biases. Next, we need to find out a type of fuzzy number to denote the fuzzy inputs, fuzzy weights and fuzzy biases, this type fuzzy number has good property so that it can be easily adept to the interval arithmetic.

Furthermore, let (x_p, T_p) is the fuzzy input-output pairs, and $T_p = (T_{p1}, T_{p2}, ..., T_pN_O)$ is N_O– dimensional fuzzy target vector corresponding to the fuzzy input vector x_p.

The cost function for the input-output pair (x_p, T_p) is obtained as

$$e_p = \sum_h e_{ph} \quad ...(17.12)$$

The cost function for the h-level sets of the fuzzy output vector O_p and the fuzzy target vector are defined as

$$e_{ph} = \sum_{k=1}^{N_O} e_{pkh} \quad ...(17.13)$$

where

$$e_{pkh} = e_{pkh}^L + e_{pkh}^U$$

$$e_{pkh}^L = h.\frac{\left([T_{pk}]_h^L - [O_{pk}]_h^L\right)^2}{2}$$

$$e_{pkh}^U = h.\frac{\left([T_{pk}]_h^U - [O_{pk}]_h^U\right)^2}{2}$$

Next section we introduce the strong L-R type fuzzy number, and put forward a FNN algorithm

based BP.

17.9.1 Strong *L-R* Representation of Fuzzy Numbers

Definition 17.4: A function, usually denoted L or R, is a reference function of fuzzy numbers if

1. $S(0) = 1$;
2. $S(x) = S(-x)$;
3. S is non-increasing on $[0, +\infty]$.

Definition 17.5: A fuzzy number M is said to be an L-R type fuzzy number if

$$\mu M(x) = \begin{cases} L\left(\dfrac{\beta - x}{a}\right) & \text{if } x \le \beta \quad a > 0 \\ R\left(\dfrac{x - \beta}{b}\right) & \text{if } x \ge \beta \quad b > 0 \end{cases} \qquad ...(17.14)$$

L is the left and R for right reference. M is the mean value of M. a and b are called left and right spreads, symbolically, we write $M = (ma\beta)_{LR}$.

Definition 17.6: This kind of fuzzy number M is said to be an strong L-R type fuzzy number if $L(1) = R(1) = 0$.

This kind of fuzzy number has properties as follows:

1. The α-cuts of every fuzzy number are closed intervals of real numbers
2. Fuzzy numbers are convex fuzzy sets.
3. Let $(\beta - x) | a = 1, x = \beta - a \equiv a$ such that, $L(\beta - a) = L(a) = 0$, same as $R(\beta + b) = R(v) = 0$, such that the support of every fuzzy number is the interval (a, v) of real numbers.

Those properties are essential for defining meaningful arithmetic operations on fuzzy numbers. Since each fuzzy set is uniquely represented by its α-cut. These are closed intervals of real numbers, arithmetic operations on fuzzy numbers can be defined in terms of arithmetic operation on closed intervals of real numbers. These operations are the cornerstone of interval analysis, which is a well-established area of classical mathematics. We will utilize them in next section to define arithmetic operations on fuzzy numbers.

The strong L-R type is an important kind of fuzzy numbers. The triangular fuzzy number (T.F.N.) is a special class of the strong L-R type fuzzy number. We can write any strong L-R type fuzzy number symbolically as $M = (a, \beta, v)_{LR}$, in other words, the strong L-R type fuzzy number can be uniquely represented by three parameters. Accordingly we can adjust three parameters of each strong L-R type fuzzy weight and fuzzy biases.

We express the strong L-R type fuzzy weight w_{ji}, w_{kj} and fuzzy biases Θ_j, Θ_k by these parameters as

$$\begin{aligned} W_{kj} &= (w_{kj}^{\alpha}, w_{kj}^{\beta}, w_{kj}^{\gamma})_{LR} \\ W_{ji} &= (w_{ji}^{\alpha}, w_{ji}^{\beta}, w_{ji}^{\gamma})_{LR} \\ \Theta_k &= (\theta_k^{\alpha}, \theta_k^{\beta}, \theta_k^{\gamma})_{LR} \end{aligned} \qquad ...(17.15)$$

$$\Theta_j = (\theta_j^{\alpha}, \theta_j^{\beta}, \theta_j^{\gamma})_{LR}$$

Let $$c_{kj} = \frac{w_{kj}^{\gamma} - w_{kj}^{\beta}}{w_{kj}^{\beta} - w_{kj}^{\alpha}},\ c_{ji} = \frac{w_{ji}^{\gamma} - w_{ji}^{\beta}}{w_{ji}^{\beta} - w_{ji}^{\alpha}},\ c_k = \frac{\theta_k^{\gamma} - \theta_k^{\beta}}{\theta_k^{\beta} - \theta_k^{\alpha}},\ \text{and}\ c_j = \frac{\theta_j^{\gamma} - \theta_j^{\beta}}{\theta_j^{\beta} - \theta_j^{\alpha}}$$

Then $$w_{kj}^{\beta} = \frac{w_{kj}^{v} + c_{kj} + w_{kj}^{\alpha}}{1 + c_{kj}},\ w_{ij}^{\beta}, \theta_k^{\beta}, \theta_k^{\beta} \text{ have same form as } w_{kj}^{\beta}.$$

We discuss how to learn the strong *L-R* type fuzzy weight $w_{kj} = (w_{kj}^{\alpha}, w_{kj}^{\beta}, w_{kj}^{v})$ between the *j*th hidden unit and the *k*th output unit. Similar to Rumelhart, we can count the quantity of adjustment for each parameter by the cost function

$$\Delta w_{kj}^{L}(t) = -\eta \frac{\partial e_{ph}}{\partial w_{kj}^{L}} + \xi \bullet \Delta w_{kj}^{L}(t-1) \qquad ...(17.16)$$

$$\Delta w_{kj}^{U}(t) = -\eta \frac{\partial e_{ph}}{\partial w_{kj}^{U}} + \xi \bullet \Delta w_{kj}^{U}(t-1) \qquad ...(17.17)$$

The derivates of above can be written as follows:

$$\frac{\partial e_{ph}}{\partial w_{kj}^{\alpha}} = \frac{\partial e_{ph}}{\partial [w_{kj}]_h^{\alpha}} \frac{\partial [w_{kj}]_h^{\alpha}}{\partial w_{kj}^{\alpha}} + \frac{\partial e_{ph}}{\partial [w_{kj}]_h^{v}} \frac{\partial [w_{kj}]_h^{v}}{\partial w_{kj}^{\alpha}} \qquad ...(17.18)$$

$$\frac{\partial e_{ph}}{\partial w_{kj}^{v}} = \frac{\partial e_{ph}}{\partial [w_{kj}]_h^{\alpha}} \frac{\partial [w_{kj}]_h^{\alpha}}{\partial w_{kj}^{v}} + \frac{\partial e_{ph}}{\partial [w_{kj}]_h^{v}} \frac{\partial [w_{kj}]_h^{v}}{\partial w_{kj}^{v}} \qquad ...(17.19)$$

Since w_{kj} is a strong *L-R* type fuzzy number, it is *h*-level and 0-level have relations as follows:

$$[w_{kj}]_h^{\alpha} = \frac{w_{kj}^{v} + c_{kj} + w_{kj}^{\alpha}}{1 + c_{kj}} - \left[\frac{(w_{kj}^{v} - w_{kj}^{\alpha})}{1 + c_{kj}}\right] L^{-1}(h) \qquad ...(17.20)$$

$$[w_{kj}]_h^{v} = \frac{w_{kj}^{v} + c_{kj} + w_{kj}^{\alpha}}{1 + c_{kj}} - \left[\frac{c_{kj}(w_{kj}^{v} - w_{kj}^{\alpha})}{1 + c_{kj}}\right] R^{-1}(\mathrm{h}) \qquad ...(17.21)$$

Therefore,

$$\frac{\partial e_{ph}}{\partial w_{kj}^{\alpha}} = \frac{\partial e_{ph}}{\partial [w_{kj}]_h^{\alpha}} \left[\frac{c_{kj}}{1 + c_{kj}} + \frac{L^{-1}(h)}{1 + c_{kj}}\right] + \frac{\partial e_{ph}}{\partial [w_{kj}]_h^{y}} \left[\frac{c_{kj}}{1 + c_{kj}} - \frac{c_{kj}}{1 + c_{kj}} R^{-1}(h)\right] \qquad ...(17.22)$$

$$\frac{\partial e_{ph}}{\partial w_{kj}^{y}} = \frac{\partial e_{ph}}{\partial [w_{kj}]_h^{\alpha}} \left[\frac{c_{kj}}{1 + c_{kj}} - \frac{L^{-1}(h)}{1 + c_{kj}}\right] + \frac{\partial e_{ph}}{\partial [w_{kj}]_h^{y}} \left[\frac{1}{1 + c_{kj}} + \frac{c_{kj}}{1 + c_{kj}} R^{-1}(h)\right] \qquad ...(17.23)$$

These relations explain how the error signals $\frac{\partial e_{ph}}{[w_{kj}]_h^\alpha}$ and $\frac{\partial e_{ph}}{[w_{kj}]_h^\alpha}$ for the *h-level*set propagate to the 0-level of the strong *L-R* type fuzzy weight W_{kj}, and then, the fuzzy weight is updated by the following rules:

$$w_{kj}^\alpha(t+1) = w_{kj}^\alpha(t) + \Delta w_{kj}^\alpha \quad ...(17.24)$$

$$w_{kj}^\gamma(t+1) = w_{kj}^\gamma(t) + \Delta w_{kj}^\gamma \quad ...(17.25)$$

We assume that n values of h (i.e., h_1, h_2, …, h_n) are used for the learning of the fuzzy neural network. In this way, the learning algorithm of the fuzzy neural network can be defined as follows:

1. Initialize the fuzzy weights and the fuzzy biases.
2. Repeat 3 for $h = h_1, h_2, ..., h_n$
3. Repeat the following procedure for $p = 1, 2, ..., m$ (m input-output pairs (x_p, T_p)).
   Forward calculation: Calculate the h–level set of the fuzzy output vector O_p corresponding to the fuzzy input vector x_p.
   Back-propagation: Adjust the fuzzy weights and the fuzzy biases using the cost function c_{ph}.
4. If a pre-specified stopping condition (etc., the total number of iterations) is not satisfied, go to 2.
   Let (x_p, T_p) is the fuzzy input-output pairs, and $T_p = (T_{p1}, T_{p2}, ..., T_pN_o)$whaere N_o–dimensional fuzzy taget vector corresponding to the fuzy input vector x_p.

17.9.2 Simulation

We consider a n-dimension classification problem. It can be described by IF-THEN rules as follows:

IF x_{p1} is A_{p1} and, …, and x_{pn} is A_{pn}.

THEN xp = $(x_{p1}, \wedge, x_{pn})$ beling to G_p.

Where $p = 1, 2, ..., k.A_{pi}$ is lingistic term, for example: “large”, “small” etc. for the convenience of computing, we assume that A_{pi} is a symmetrical strong *L-R* type fuzzy number, that is to say, $L = R + \max(0, 1 - |x|^2)$.

We can solve the above problem by using the fuzzy neural network we discussed. So we note the fuzzy input as $A_p = (A_{p1}, ..., A_{pn})$, and the target ouput T_p can be defined as follows:

$$T_p = \begin{cases} 1 & \text{if } A_p \in \text{class } 1 \\ 0 & \text{if } A_p \in \text{class } 2 \end{cases} \quad ...(17.26)$$

according to the target output T and the real output O.

We define the error function:

$$e_{ph} = \max \{(t_p - o_p)^2/2 | O_p \in [y_p]_h\} \quad ...(17.27)$$

We should train this network in order to make e_{ph} be minimum. It is easy to know that the error function become the classical error function $e = \sum_{p=1}^{k} (t_p - o_p)^2 / 2$ in BP algorithm when input vector A_p

and y_p are real numbers. We train the fuzzy neural network with h-levelsets (h = 0.2, 0.4, 0.6, 0.8), the error function of the pth pair is:

$$e_{ph} = \sum_{p} h \bullet \max \{(t_p - o_p)^2 / 2 | O_p \in [y_p]h\} \quad ...(17.28)$$

The result of the trained fuzzy neural network is shown in Fig. 17.8. We assume A_1-A_4 belong to class1, and A_5-A_8 belong to class 2. Using the proposed learning algorithm, we get a satisfied curve after 300 echoes.

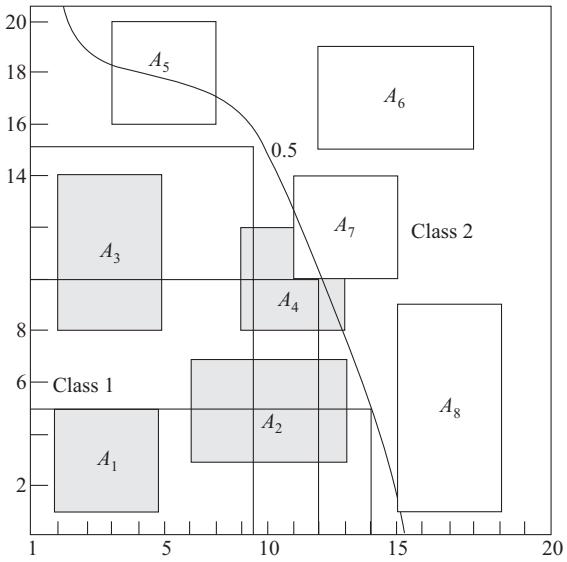

Fig. 17.8 The result of fuzzy classification

17.10 ADAPTIVE NEURO-FUZZY INFERENCE SYSTEM (ANFIS)

A fuzzy system can be considered to be a parameterized nonlinear map, called f. Let's write here explicitly the expression of f.

$$f(x) = \frac{\sum_{l=1}^{m} y^l \left(\prod_{i=1}^{n} \mu_{A_i^l}(x_i) \right)}{\sum_{l=1}^{m} \left(\prod_{i=1}^{n} \mu_{A_i^l}(x_i) \right)} \quad ... (17.29)$$

where y' is a place of output singleton if Mamdani reasoning is applied or a constant if Sugeno reasoning is applied. The membership function $\mu_{A_i^l}(x_i)$ corresponds to the input $x = (x_1, ..., x_n)$ of the rule l. The ***and*** connective in the premise is carried out by a product and defuzzification by the center-of-gravity method.

This can be further written as

$$f(x) = \sum_{i=1}^{m} w_i b_i(x) \qquad ...(17.30)$$

where $w_i = y'$

$$b_j(x) = \frac{\prod_{i=1}^{n} \mu_{A_i^l}(x_i)}{\sum_{l=1}^{m} \prod_{i=1}^{n} \mu_{A_i^l}(x_i)}$$

If F is a continuous, nonlinear map on a compact set, then f can approximate F to any desired accuracy, i.e.

$$F \approx f_{FS} \qquad ...(17.31)$$

Well-known theorems from Approximation theory for polynomials, can be extended to fuzzy systems (e.g. Wang: A course in fuzzy systems and control, Prentice Hall, 1997).

The following theorems are found in R.F. Curtain and A.J. Pritchard: *Functional Analysis in Modern Applied Mathematics* as Corollaries of Orthogonal Projection Theorem.

Theorem 17.5: Let F be a bounded function on $[a, b]$, and $E = (x_1, ..., x_k)$ a set of points in $[a, b]$. Then there exists the least squares polynomial of degree $\leq n$, p_n^k which *minimizes*

$$\sum_{i=1}^{k} |F(x_i) - p(x_i)|^2 \qquad ...(17.32)$$

over all polynomials of degree $\leq n$.

Theorem 17.6: If $F \in C[a, b]$, then for any $n \geq 0$, there exists a best approximating polynomial π_n of degree $\leq n$ such that

$$\|F - \pi_n\|_\infty \leq \|F = p\|_\infty \qquad ...(17.33)$$

over all polynomials p of degree $\leq n$.

Remark: The message of Theorem 2 is that polynomials are dense in the space of continuous functions in Hilbert space. The same can also be said of trigonometric functions. We can also consider the simpler problem of approximating at finitely many points.

Theorem 17.7: If F is a bounded function on $[a, b]$, and $E = (x_1, ..., x_k)$ a set of points in $[a, b]$. Then there exists a best approximating polynomial π_n^k of degree $\leq n$, which *minimizes*

$$\max_{0 \leq i \leq k} |F(x_i) - p(x_i)| \qquad ...(17.34)$$

over all polynomials of degree $\leq n$.

17.10.1 ANFIS Structure

Consider a Sugeno type of fuzzy system having the rule base

1. If x is A_1 and y is B_1, then $f_1 = p_1 x + q_1 y + r_1$
2. If x is A_2 and y is B_2, then $f_2 = p_2 x + q_2 y + r_2$

Let the membership functions of fuzzy sets A_i, B_i, $i = 1, 2$, be μ_{A_i}, μ_{B_i}.

In evaluating the rules, choose *product* for T-norm (logical *and*).

1. Evaluating the rule premises results in

$$w_i = \mu_{A_i}(x)\ \mu_{B_i}(x), \qquad i = 1, 2 \qquad ...(17.35)$$

2. Evaluating the implication and the rule consequences gives

$$f(x, y) = \frac{w_1(x, y)\, f_1(x, y) + w_2(x, y)\, f_2(x, y)}{w_1(x, y) + w_2(x, y)} \qquad ...(17.36)$$

Or leaving the arguments out

$$f = \frac{w_1 f_1 + w_2 f_2}{w_1 + w_2} \qquad ...(17.37)$$

This can be separated to phases by first defining

$$\overline{w}_i = \frac{w_i}{w_1 + w_2} \qquad ...(17.38)$$

Then f can be written as

$$f = \overline{w}_1 f_1 + \overline{w}_2 f_2 \qquad ...(17.39)$$

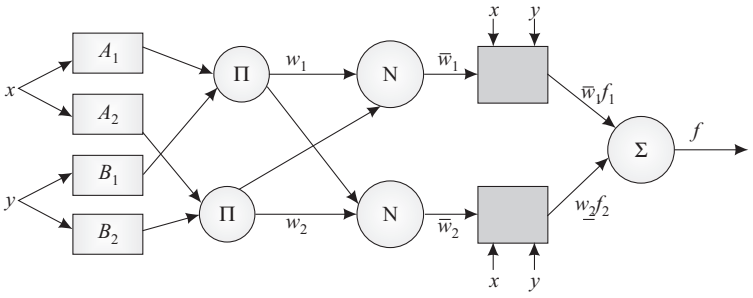

Fig. 17.9 ANFIS structure

All computations can be presented in a diagram form.

QUESTION BANK

1. What are the different types of hybrid systems? Explain them schematically.
2. Explain the use of fuzzy logic in neural networks to improve the learning ability.
3. Define the following:
   - Hybrid neural network
   - Regular fuzzy neural network
   - Hybrid fuzzy neural network
4. Explain the role of neural networks as pre-processor or post-processor of data.
5. What is committee of networks? Explain.
6. Describe FNN architecture based on back propagation.
7. Describe adaptive neuro-fuzzy inference system.

REFERENCES

1. B. Kosko, *Neural Network and Fuzzy Systems*, Prentice Hall, Englewood, NJ, 1991.
2. G.A. Carpenter and S. Grossberg, The art of adaptive pattern recognition by a self organizing neural network, *IEEE Computer Magazine*, Vol. 21, No. 3, pp. 77-88.
3. S. Haykin, *Neural Networks*: *A Comprehensive Foundation*, IEEE computer society press, Macmillan, NY.
4. S. Kartalopoulos, *Understanding Neural Networks and Fuzzy Logic*, IEEE press, NY.
5. H.M. Lee and B.H. Liu, Fuzzy BP: a neural network model with fuzzy inference, Proceedings of International conference on artificial neural networks, pp. 1583-1588, 1994.

CHAPTER 18

Hybrid Fuzzy Neural Networks Applications

18.1 INTRODUCTION

The hybrid fuzzy neural networks have a tremendous potential to solve engineering problems. It is improper to expect that if the individual technologies are good then hybridization of technologies should turn out to be even better. Hybridization is performed for the purpose of investing better methods of problem solving.

18.2 TOOL BREAKAGE MONITORING SYSTEM FOR END MILLING

In recent years, global competition in industry has led to exploration of new means of more efficient production. In particular, flexible manufacturing systems (FMS) have been investigated as a tool for raising manufacturing productivity and product quality while decreasing production costs. One type of FMS is the Unmanned Flexible Manufacturing System (UFMS), which has received a great deal of attention because it replaces human operators with robotic counterparts in manufacturing and assembly cells. In addition to performing the same function as the FMS, the UFMS reduces direct labour costs and prevents personal oversights.

However, since human operators are absent in these systems, electronic sensors associated with a decision-making system must monitor the process. The decision-making system analyzes information provided by sensors to make appropriate control actions. In order to ensure efficiency within the system, monitoring equipment and algorithms for the adaptation of the manufacturing process must be executed accurately (Altintas, Yellowley, and Tlusty, 1988).

To apply the UFMS effectively, manufacturers must confirm that the tool is in good condition in process; therefore, automatic and rapid detection of tool breakage is central to successful UFMS operation. By themselves, computer numerical control (CNC) machines are not typically capable of tool breakage detection. Since CNC machines cannot detect tool conditions, they cannot halt the process if the tool becomes damaged. Materials costs increase and product quality suffers if a broken tool is used in production. To reduce costs of materials and prevent damaged tools from negatively affecting production, a detecting technology for unexpected tool breakage is needed (Lan and Naerheim, 1986).

An in-process tool breakage detection system was developed in an end milling operation with cutting force and machining parameters of spindle speed, feed rate, and depth of cut selected as input factors. The neural networks approach was employed as the decision-making system that judges tool conditions.

The common method of detecting tool breakage in process involves force signals resultant from tool processes on raw materials. A dynamometer sensor is the main device used to measure force signals in different machining operations. Lan and Naerheim (1986) proposed a time series auto regression (AR) model of force signals to detect tool breakage. The time-series-based tooth period model technique (TPMT), which used the fast a posterior error sequential technique (FAEST), was applied by Tansel and McLaughlin (1993) to detect tool breakage in milling. Tarng and Lee (1993) proposed using the average and median force of each tooth in the milling operation. Measured by sensors, the average and median forces of each tooth were used as input information. An appropriate threshold was built to analyze information and detect tool conditions.

Jemielniak (1992) proposed that sudden changes in the average level of force signals could be due to catastrophic tool failure (CTF) in turning operations. In this study, analyzing force signals and determining amplitude fluctuations allowed on-line tool breakage detection. Zhang, Han, and Chen (1995) used a telemeter technique, a battery-powered sensing force/torque cutter holder mounted on the spindle head with the transmitter, to measure force in milling operations.

The application of neural networks and fuzzy logic in detecting tool breakage has also been studied in recent years. Tae and Dong (1992) developed a fuzzy pattern recognition technique and a time-series AR model to detect tool wear in turning operations. The variation of dynamic cutting force was used to construct the fuzzy dispersion pattern needed to distinguish tool conditions. Ko, Cho, and Jung (1994) introduced an unsupervised self-organized neural network combined with an adaptive time-series AR modeling algorithm to monitor tool breakage in milling operations. The machining parameters and average peak force were used to build the AR model and neural network.

Chen and Black (1997) also introduced a fuzzy-nets system to distinguish tool conditions in milling operations. The fuzzy-nets system was designed to build the rule-bank and solve conflicting rules with a computer. Variance of adjacent peak force was selected as an input parameter to train the system and build a rule-bank for detecting tool breakage.

18.2.1 Methodology: Force Signals in the End Milling Cutting Process

Milling is a fundamental machining process in the operation of CNC machines, and milling operations can be of two varieties: peripheral and face milling. Milling is an interrupted cutting process, meaning that each cutting tooth moving in the same direction generates a cyclic cutting force ranging from zero to maximum force, and back to zero. This cyclic force is graphed as a series of peaks. The principle of cutting force can be further defined as resultant force, F_R, generated in x and y directions. The resultant force, F_{ri}, generated from x and y directions, was used in this experiment expressed as:

$$F_{r_i} = \sqrt{F_{x_i}^2 + F_{y_i}^2} \qquad ...(18.1)$$

Where

F_{r_i} is the resultant force of point I

F_{x_i} is the force in X direction of point I

F_{y_i} is the force in Y direction of point i.

Tool conditions and machining parameters affect the magnitude of resultant force; therefore, if the tool condition is good, the peak measurement of each tooth's force should be roughly the same from tooth to tooth during one revolution of the cutting process. Comparatively, if a tooth is broken, it generates a smaller peak force because it carries a smaller chip load. As a result, the tooth that follows a broken tooth generates a higher peak force as it extracts the chip that the broken tool could not.

Applying the force principle, two main differences can be used to detect tool breakage:

1. maximum peak force in each revolution should differ between good and broken tools, and maximum peak force of a broken tool must be larger than that of a good tool;
2. Maximum variance force of adjacent peaks should differ between good and broken tools, and maximum variance force of adjacent peaks of broken tools must be larger than in undamaged tools.

Figure 18. 1 illustrates the diagram of undamaged and broken tools. In this work, an in-process tool breakage detection system was developed in an end milling operation. The cutting force and machining parameters, such as spindle speed, feed rate, and depth of cut, were selected as input factors. Finally, the neural networks approach was used as a decision-making system using input from sensors to judge tool conditions.

18.2.2 Neural Networks

In this work, a back propagation neural network (BPNN) was chosen as the decision-making system because it is the most representative and commonly used algorithm. It is relatively easy to apply, has been proven effective in dealing with this kind of task, and has also proven successful in practical applications.

Back propagation is intended for training layered (i.e., nodes are grouped in layers), feed forward (i.e., the arcs joining nodes are unidirectional, and there are no cycles) nets, as shown in Figure. 18.1. This approach involves supervised learning, which requires a 'teacher' that knows the correct output for any input, and uses gradient descent on the error provided by the teacher to train the weights. As the weights of the neural network were obtained, the prediction function was achieved via the weight information. The propagation rule, also called a summation or aggregated function, was used to combine or aggregate inputs passing through the connections from other neurons. It can be expressed as

$$S_i = \sum a_i W_{ji} - a_0 W_{jo} \qquad ...(18.2)$$

$$S_k = \sum a_k W_{jk} - a_0 W_{jo} \qquad ...(18.3)$$

where, i is an input neuron, j is a hidden neuron, and k is an output neuron, W_{ji} and W_{kj} denote weight from input to hidden neuron, and from hidden to output neuron respectively, while a_0 represents the bias, usually 1, and W_{ji} and W_{ji} are the weight of bias.

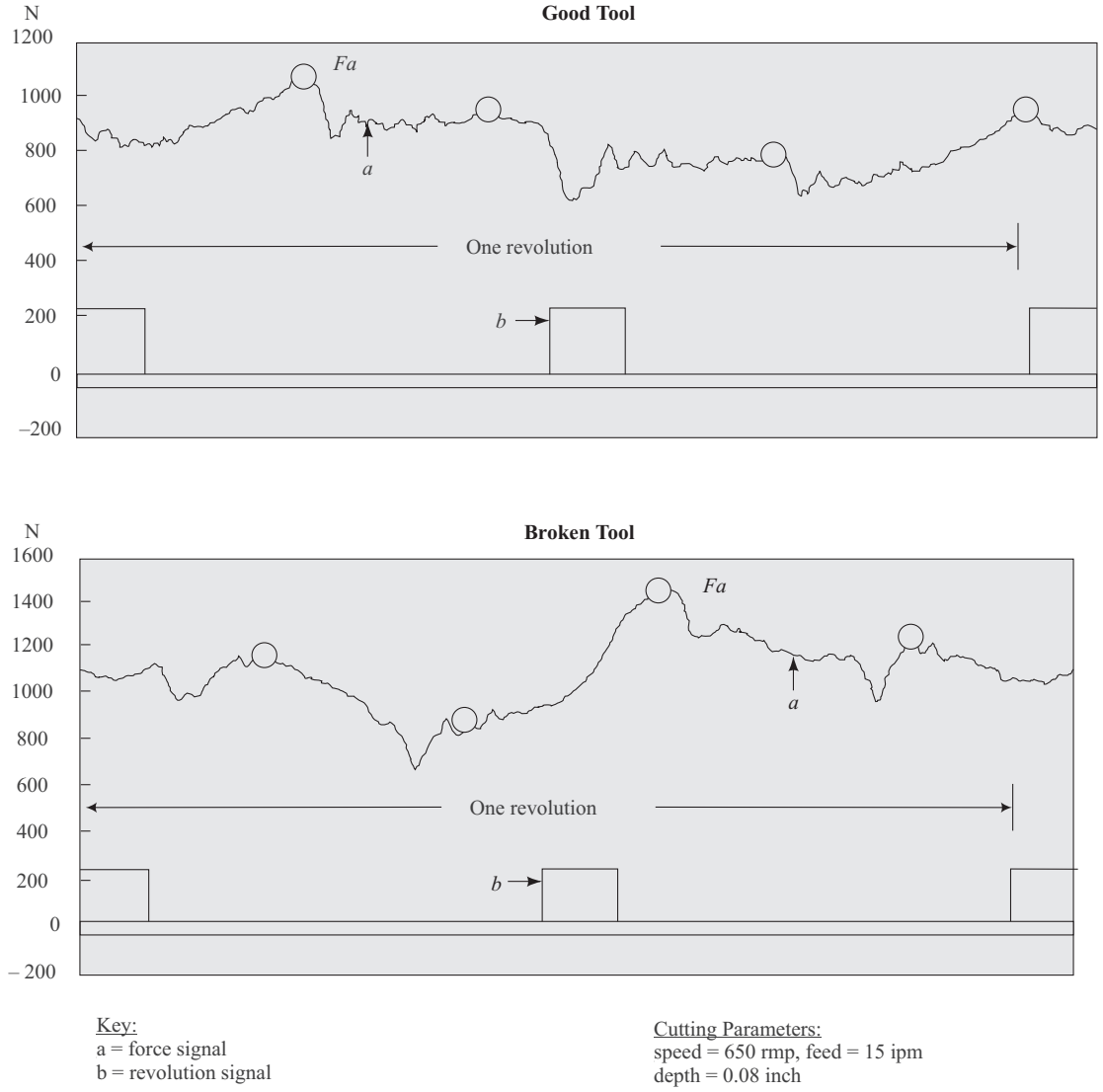

Fig. 18.1 The amplitude of cutting force of a good and broken tool.

The transfer function, also called the 'output' or 'squashing' function, is used to produce output based on level of activation. Many different transfer functions can be used to transfer data, and one is called the Sigmoid Function, expressed as:

$$O_y = \frac{1}{1 + e^{-ay}} \qquad ...(18.4)$$

where ay is a function of S_j and S_k respectively.

Comparing actual output of neural networks to desired output, the process is repeated until the error percentage falls into a reasonable range.

18.2.3 Experimental Design and System Development Experimental Design

This experiment employed a CNC vertical machining center. A dynamometer was mounted on the table to measure cutting force. A proximity sensor was built near the spindle to confirm data in each revolution. Four ¾-inch doubt-end four-flute high-speed steel cutters were used. In each cutter, one side of the tool was in proper working order and the other side was broken. The experimental setup was shown in Fig. 18.2. The broken side of the tool possessed varying degrees of breakage (Fig. 18.3). The cutting parameters were set as:

1. Five level of spindle speed (740, 500, 550, 600, and 650 revolution per minute),
2. Five level of feed rate (6, 12, 18 and 24 inch per minute), and
3. Five levels of depth of cut (0.06, 0.07, 0.08, 0.09 and 0.1 inches).

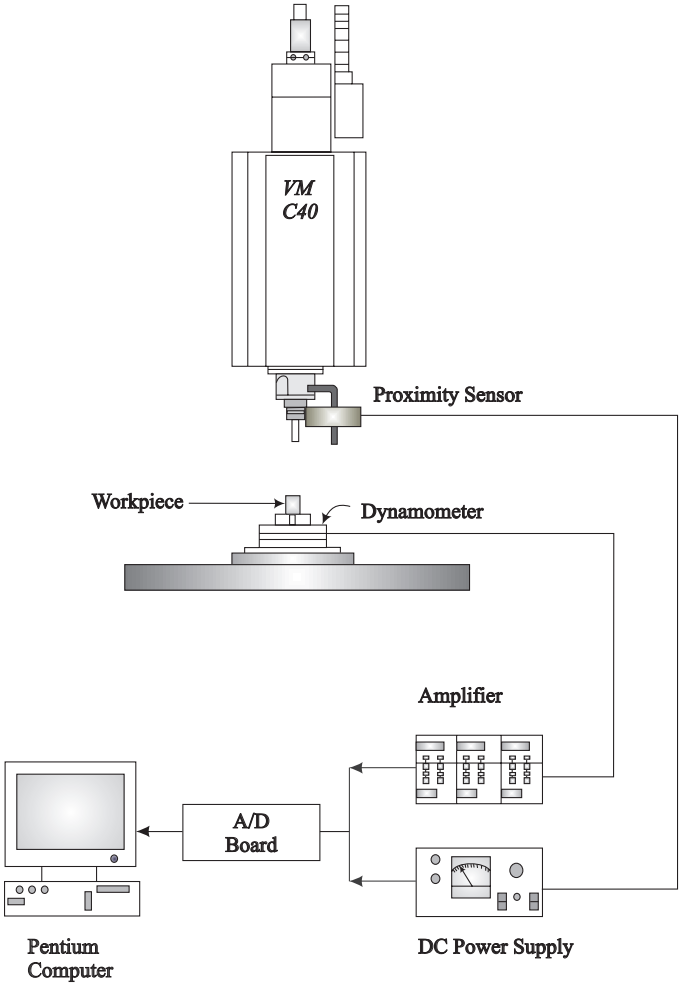

Fig. 18.2 The experimental setup.

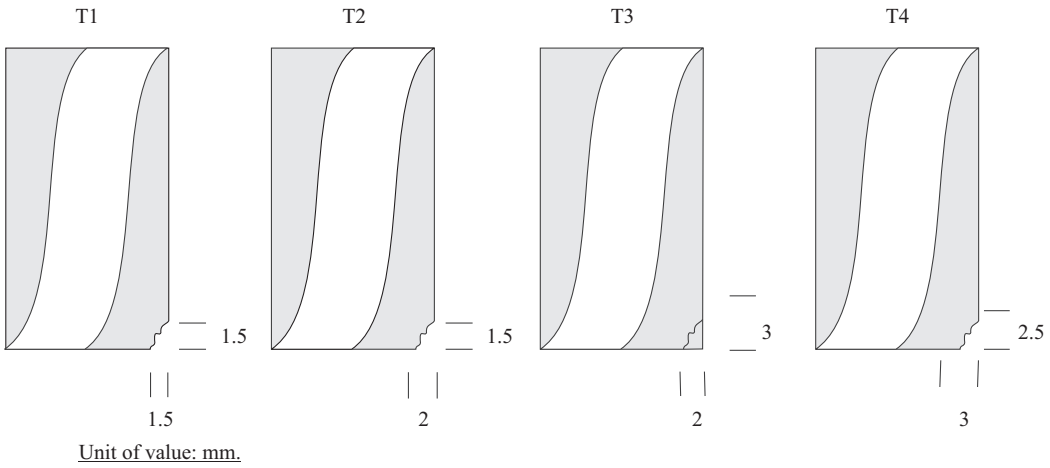

Fig. 18.3 Diagram of broken tool.

The cutters used to execute the experiment were selected randomly. Cutting force was measured in voltage by the Charge Amplifier and transformed to Newtons (*N*) via computer.

18.2.4 Neural Network-BP System Development

To develop back propagation of neural networks as a decision-making system, MATLAB software was applied to analyze data. Seven steps were conducted. In step one, prediction factors were determined in order to perform the training process. Step 2 was necessary to analyze differences between scaling data and unscaling data. Step 3 dealt with separating data into training and testing categories. From steps 4 through 6, parameters were developed for the training process, including the hidden layer/hidden neuron, learning rate, and momentum factor. Finally, in step 7, information from the training process was used to predict tool conditions.

Step 1. Determine the factors

Five input neurons were used for tool breakage prediction data:

1. Spindle speed;
2. Feed rate;
3. Depth of cut;
4. Maximum peak force; and
5. Maximum variance of adjacent peak force.

Output neurons were either (1) Good, or (2) Broken.

Three hundred data points were used in this work. Good tools collected half of these and broken tools collected the rest, and all data were randomized using MS Excel software.

Step 2. Analyze unscaling and scaling data

In order to avoid experimental errors resulting from bigger values of some data sets, some pre-processing was needed to obtain good training and prediction results. Since histograms of all data sets

were uniform or normal distributions, the Simple Linear Mapping method was employed for scaling. To compare the difference between two sample sizes, some parameters were first set and fixed.

The number of hidden neurons was set at 4, the learning rate was set at 1, and the momentum item was 0.5. The number of training cycles was 2000, and the testing period was 5. Table 18.1 shows the comparison of the difference between scaling and unscaling data. As one can see, errors in scaling data are smaller than in unscaling data.

Table 18.1 Difference between scaling and non-scaling

	Hidden layer	*Hidden neuron*	*Learn rate*	*Momentum factor*	*Train Error*	*Testing Error*	*RMS error of training*	*RMS error of testing*
Unscale	1	4	1	0.5	0.505	0.550	0.508	0.515
Scale	1	4	1	0.5	0.040	0.160	0.197	0.388

Step 3. Impact of the ratio of training and testing data

The 300 original 300 data records were randomized and separated into three groups. The first group had 200 training data and 100 (200 × 100) testing data, the second had 225 training and 75 (225 × 75) testing data, and the third had 250 training and 50 testing (250 × 50) data. Table 18.2 shows the Back Propagation Neural Network (BPNN) with different sample sizes of training and testing data. The last four columns of Table 18.2 show training and testing errors.

The training, testing, and RMS (root mean square) errors of training of the second group were smaller than in other groups. The RMS errors of testing data of the second group sample were larger than in the first; however, the RMS errors of each sample size were very similar. If samples had similar error percentages, the sample with the largest training sample size was selected because it provided sufficient information to predict testing data. From the experimental design, the ideal ratio between training and testing data was 3:1 for neural networks. The 225 × 75 sample size was employed in this analysis.

Table 18.2: Different sample size of training and testing data

*Tra*Tes*	*Hidden layer*	*Hidden neuron*	*Learn rate*	*Momentum factor*	*Train Error*	*Testing Error*	*RMS error of training*	*RMS error of testing*
200* 100	1	4	1	0.5	0.040	0.160	0.197	0.388
225* 75	1	4	1	0.5	0.036	0.093	0.185	0.298
250* 50	1	4	1	0.5	0.044	0.106	0.204	0.285

Step 4. Impact of the hidden layer and hidden neuron

In the beginning, the number of hidden neurons was set at 5, and the hidden layer was set at 1. Different hidden neurons and layers were tested to determine which values would lead to the smallest error percentage. To this end, the hidden neurons were set at 4 and 6, and the hidden layers were set at 1 and 2. Table 18.3 shows the BPN with a different number of hidden neurons and layers. According to this data, the percentage error of the trial with 4 hidden neurons and 1 hidden layer was less than it was in all

other trials. Thus, the configuration contained in the 4 hidden neuron/1 hidden layer experiment was chosen because it led to the best results. The formula, (input neurons + output neurons)/2, was useful for determining the number of hidden neurons at the beginning.

Table 18.3: Different number of hidden neurons and layers

Neuron in layer-1	*Neuron in layer-2*	*Learn rate*	*Momentum factor*	*Train Error*	*Testing Error*	*RMS error of training*	*RMS error of testing*
3	~	1	0.5	0.080	0.200	0.256	0.410
4	~	1	0.5	0.036	0.093	0.185	0.298
5	~	1	0.5	0.049	0.093	0.193	0.288
3	3	1	0.5	0.267	0.320	0.338	0.375
4	2	1	0.5	0.489	0.453	0.512	0.504
4	3	1	0.5	0.316	0.333	0.407	0.425
4	4	1	0.5	0.164	0.227	0.362	0.414

Step 5. Impact of the learning rate

This step was necessary to determine the optimal learning rate. The initial learning rate was 1. Three additional learning rates, 0.5, 2, and 10, were used to compare with the initial. Table 18.4 shows the BPN with different learning rate values. Table 18.4 shows that the error percentage of the learning rates of 0.5 and 1 were the same, in addition to being lower than all other learning rates. To achieve the objective of finding the smallest error percentage, the learning rate of 1 was used, because the software originally recommended that value.

Table 18.4 Different learning rate values

Hidden layer	*Hidden neuron*	*Learn rate*	*Momentum factor*	*Train Error*	*Testing Error*	*RMS error of training*	*RMS error of testing*
1	4	0.5	0.5	0.036	0.093	0.185	0.298
1	4	1	0.5	0.036	0.093	0.185	0.298
1	4	2	0.5	0.116	0.133	0.306	0.319
1	4	10	0.5	0.111	0.133	0.286	0.317

Step 6. Impact of the momentum factor

The final step of data analysis was to change the value of the momentum item to obtain the configuration leading to the lowest error percentage. The initial value of the momentum item was 0.5. Another three values, 0.3, 0.6, and 0.8, were selected to compare with the initial value. Table18.5 shows the BPN with different values for the momentum item. Table 18.5 shows that the percentage of errors of momentum items of 0.3 and 0.5 are the same, and smaller than all others. To achieve the smallest error percentage, the 0.5 momentum item was used, because the software originally recommended that value.

Step 7. Prediction

After completing analysis and obtaining information about weight and input factors, equations to predict tool conditions were constructed. The variables a_1, a_2, …., and a_5 represent 5 input factors, maximum peak force, spindle speed, feed rate, depth of cut, and maximum variance of adjacent peak force, respectively. By application of equations (18.5), the weighted value of hidden factors a_{h1}, a_{h2}, a_{h3}, a_{h4} can be expressed as:

Table 18.5 Percent error of momentum items

Hidden layer	Hidden neuron	Learn rate	Momentum factor	Train Error	Testing Error	RMS error of training	RMS error of testing
1	4	1	0.3	0.036	0.093	0.185	0.298
1	4	1	0.5	0.036	0.093	0.185	0.298
1	4	1	0.6	0.049	0.093	0.212	0.288
1	4	1	0.8	0.116	0.133	0.306	0.322

$$a_{h1} = \frac{1}{1+\exp^{-[a_1\times(4.652)+a_2\times(0.448)+a_3\times(0.947)+a_4\times(-25.237)+a_5\times(0.853)-(-0.221)]}} \quad ...(18.5)$$

$$a_{h2} = \frac{1}{1+\exp^{-[a_1\times(40.457)+a_2\times(39.421)+a_3\times(-15.261)+a_4\times(7.317)+a_5\times(-21.054)-(44.505)]}} \quad ...(18.6)$$

$$a_{h3} = \frac{1}{1+\exp^{-[a_1\times(-10.224)+a_2\times(-3.444)+a_3\times(24.252)+a_4\times(3.449)+a_5\times(4.215)-(-1.289)]}} \quad ...(18.7)$$

$$a_{h4} = \frac{1}{1+\exp^{-[a_1\times(1.321)+a_2\times(-24.736)+a_3\times(0.202)+a_4\times(0.79)+a_5\times(0.015)-(0.829)]}} \quad ...(18.8)$$

$$a_{out1} = \frac{1}{1+\exp^{-[a_{h1}\times(11.697)+a_{h2}\times(16.977)+a_{h3}\times(12.295)+a_{h4}\times(11.807)-(2.945)]}} \quad ...(18.9)$$

$$a_{out2} = \frac{1}{1+\exp^{-[a_{h1}\times(-11.697)+a_{h2}\times(-16.977)+a_{h3}\times(-12.295)+a_{h4}\times(-11.807)-(-2.945)]}} \quad ...(18.10)$$

Finally, the output information was used to judge the tool conditions:

If $a_{out1} > a_{out2}$, then the tool condition is used

If $a_{out1} < a_{out2}$, then the tool is broken.

18.2.5 Findings and Conclusions

To operate the UFMS successfully, in-process sensing techniques that relate to rapid-response decision-making systems were required. In this research, a neural networks model was developed to judge cutting force for accurate in-process tool breakage detection in milling operations. The neural networks were capable of detecting tool conditions accurately and in process. The accuracy of training data was 96.4%, and the accuracy of testing data was 90.7%. Partial results of training and testing data are shown in Tables 18.6 and Table 18.7.

Table 18.6 Partial results of training data

Tool	*Input factors*				*Output factors*			*Prediction*
Conditions	a_1	a_2	a_3	a_4	a_5	a_{out1}	a_{out2}	
Good	904.5	600	12	0.09	45.8	1	0	Good
Broken	1220.54	600	12	0.09	954.04	0	1	Broken
Good	634.64	550	10	0.06	274.64	0.98	0.02	Good
Broken	780.14	550	10	0.06	368.64	0.03	0.97	Broken
Good	674.36	650	15	0.06	248.64	1	0	Good
Broken	847.06	650	15	0.06	537.54	.02	0.98	Broken
Good	1239.4	500	18	0.07	225.2	1	0	Good
Broken	1677.92	500	18	0.07	1,159.64	0	1	Broken
Good	1413.76	450	15	0.08	300.56	0	1	Broken
Broken	1861.72	450	15	0.08	1,250.92	0.01	0.99	Broken

Table 18.7 Partial results of testing data

Tool	*Input factors*				*Output factors*			*Prediction*
Conditions	a_1	a_2	a_3	a_4	a_5	a_{out1}	a_{out2}	
Good	711.94	550	15	0.06	177.02	1	0	Good
Broken	1296.56	550	15	0.06	481.66	0.01	0.99	Broken
Good	723.32	550	12	0.07	311.52	0.35	0.65	Broken
Broken	1215.96	550	12	0.07	1,042.50	0	1	Good
Good	1084.32	550	18	0.07	192.5	0.99	0.01	Broken
Broken	1542.92	550	18	0.07	1,303.92	0	1	Good
Good	1024.46	600	18	0.07	172.22	0.98	0.02	Broken
Broken	1253.28	600	18	0.07	580.3	0.03	0.97	Broken
Good	1507.18	450	20	0.08	550.06	0.98	0.02	Broken
Broken	1876.74	450	20	0.08	1,062.02	0	1	Broken

The weights of hidden factors and output factors were generated from pre-trained neural networks, and a program was written to process these weights in order to respond to the tool conditions. Therefore, the in-process detection system demonstrated a very short response-time to tool conditions. Since tool conditions could be monitored in a real-time situation, the broken tool could be replaced immediately to prevent damage to the machine and mis-machining of the product. However, since the weights were obtained from the pre-trained process, they were fixed when they were put into the detection program. Therefore, the whole system does not have the adaptive ability to feed back information into the system.

In this work, depth of cut was employed as one input factor. However, in actual industrial environments, the surface of work materials is often uneven, implying that the depth of cut set in the computer might differ from that used to cut the workpiece. Under the circumstances, the neural networks might generate a wrong decision and misjudge the tool conditions due to fluctuating depths of cut across machining.

18.3 CONTROL OF COMBUSTION

Beside the economical and environmental advantages, there are several difficulties with burning bio fuels and municipal wastes. The combustion of those fuels or fuel-mixtures has different properties compared to the conventional fuels (coal, gas, and oil). Bio fuels and municipal wastes are very inhomogeneous. The properties (heat value, moisture content, homogeneity, density, mix ability) may vary in a large range. It causes non-steady, agitated combustion conditions; even if steady fuel feed volume is maintained, leading to increase in the emission level and variation of the generated heat flow. Those property variations are not predictable or directly measurable, only their effects on the combustion, on the steam generation and on the power production can be observed through the O_2 content of the flue gas.

This topic presents an ANFIS system, which determines the amount of fuel fed to the combustion chamber. Combined with a stoichiometric model, it predicts the flue gas properties, including the O_2 content.

18.3.1 Adaptive Neuro-Fuzzy Inference System

Fuzzy Logic Controllers (FLC) has played an important role in the design and enhancement of a vast number of applications. The proper selection of the number, the type and the parameter of the fuzzy membership functions and rules are crucial for achieving the desired.

Adaptive Neuro-Fuzzy Inference Systems are fuzzy Sugeno models put in the framework of adaptive systems to facilitate learning and adaptation. Such framework makes FLC more systematic and less relying on expert knowledge. To present the ANFIS architecture, let us consider two-fuzzy rules based on a first order Sugeno model:

Rule 1: if (x is A_1) and (y is B_1), then

($f_1 = p_1x + q_1y + r_1$)

Rule 2: if (x is A_2) and (y is B_2), then

($f_2 = p_2x + q_2y + r_2$)

One possible ANFIS architecture to implement these two rules is shown in Fig. 18.4. Note that a circle indicates a fixed node whereas a square indicates an adaptive node (the parameters are changed during training). In the following presentation O_{Li} denotes the output of node i in a layer L.

Layer 1: All the nodes in this layer are adaptive nodes, I is the degree of the membership of the input to the fuzzy membership function (MF) represented by the node:

$$O_{1,i} = \mu_{Ai}(x) i = 1, 2$$

$$O_{1,i} = \mu_{Bi-2}(y) i = 3, 4$$

A_i and B_i can be any appropriate fuzzy sets in parameter form. For example, if bell MF is used then,

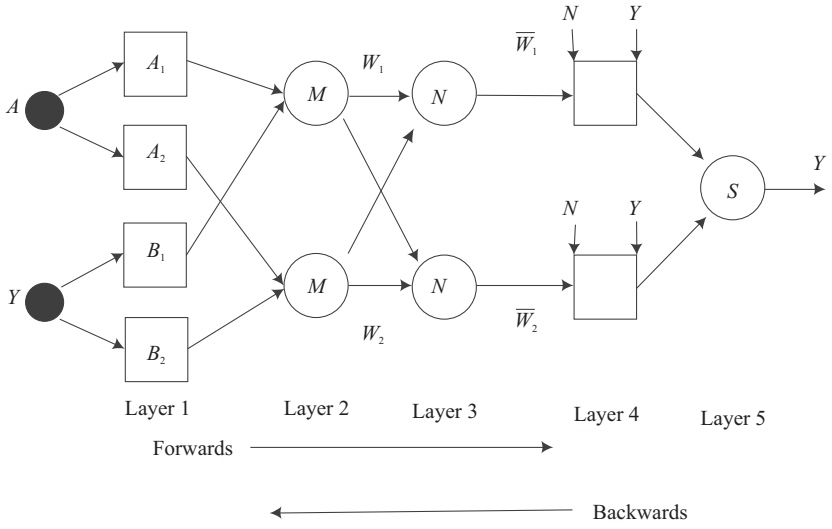

Fig. 18.4 ANFIS.

$$\mu_{Ai}(x) = \frac{1}{1 + \left[\left(\frac{x - c_i}{a_i}\right)^2\right]^{b_i}} \quad i = 1, 2 \qquad ...(18.11)$$

where a_i, b_i and c_i are the parameters for the *MF*.

Layer 2: The nodes in this layer are fixed (not adaptive). These are labelled *M* to indicate that they play the role of a simple multiplier. The outputs of these nodes are given by:

$$O_{2,i} = \mu_{Ai}(x)\, \mu_{Bi}(y)\, i = 1, 2 \qquad ...(18.12)$$

The output of each node is this layer represents the firing strength of the rule.

Layer 3: Nodes in this layer are also fixed nodes. These are labelled *N* to indicate that these perform a normalization of the firing strength from previous layer. The output of each node in this layer is given by:

$$O_{3,i} = \overline{w}_i = \frac{w_i}{w_1 + w_2} \quad i = 1, 2. \qquad ...(18.13)$$

Layer 4: All the nodes in this layer are adaptive nodes. The output of each node is simply the product of the normalized firing strength and a first order polynomial:

$$O_{4,i} = \overline{w}_i f_i = \overline{w}_i (p_i x + q_i y + r_i) \quad i = 1, 2 \qquad ...(18.14)$$

where p_i , q_i and r_i are design parameters (consequent parameter since they deal with the then-part of the fuzzy rule)

Layer 5: This layer has only one node labelled *S* to indicate that is performs the function of a simple summer. The output of this single node is given by:

$$O_{i,5} = f = \sum_i \overline{w}_i f_i = \frac{\sum_i w_i f_i}{\sum_i w_i} \quad i = 1, 2 \qquad ...(18.15)$$

The ANFIS architecture is not unique. Some layers can be combined and still produce the same output. In this ANFIS architecture, there are two adaptive layers (12, 15). Layer 1 has three modifiable parameters (a_i, b_i and c_i) pertaining to the input MFs [13]. These parameters are called *premise* parameters. Layer 4 has also three modifiable parameters (p_i, q_i and $_{ri)}$ pertaining to the first order polynomial. These parameters are called *consequent* parameters.

18.3.2 Learning Method of ANFIS

The task of training algorithm for this architecture is tuning all the modifiable parameters to make the ANFIS output match the training data [14]. Note here that a_i, b_i and c_i describe the sigma, slope and the center of the bell MF's, respectively. If these parameters are fixed, the output of the network becomes:

$$\begin{aligned} f &= \frac{w_1}{w_1 + w_2} f_1 + \frac{w_2}{w_2 + w_2} f_2 \\ &= \overline{w}_1 f_1 + \overline{w}_2 f_2 \qquad ...(18.16) \\ &= \overline{w}_1 (p_1 x + q_1 y + r_1) + \overline{w}_2 (p_2 x + q_2 y + r_2) \\ &= (\overline{w}_1 x) p_1 + (\overline{w}_1 y)\, q_1 + (\overline{w}_1) r_{1+} (\overline{w}_2 x)\, p_2 + (\overline{w}_2 y)\, q_2 + (\overline{w}_2)\, r_2 \end{aligned}$$

This is a linear combination of the modifiable parameters. For this observation, we can divide the parameter set *S* into two sets:

$$S = S_1 \oplus S_2 \qquad ...(18.17)$$

S = set of total parameters

S_1 = set of premise (nonlinear) parameters

S_2 = set of consequent (linear) parameters

$\oplus$ = direct sum

For the forward path (see Fig. 1), we can apply least square method to identify the consequent parameters. Now for a given set of values of S_1, we can plug training data and obtain a matrix equation:

$$A\Theta = y \qquad ...(18.18)$$

where Θ contains the unknown parameters in S_2. This is a linear square problem, and the solution for $\grave{E}$, which is minimizes 2 *A y È* - , is the least square estimator:

$$\Theta^* = (A^T A)^{-1} A^T y \qquad ...(18.19)$$

we can use also recursive least square estimator in case of on-line training.

For the backward path (Fig. 18.4), the error signals propagate backward. The premise parameters are updated by descent method [15], through minimizing the overall quadratic cost function

$$J(\Theta) = \frac{1}{2}\sum_{i=1}^{N}[y(k) - \hat{y}(k, \Theta)]^2 \qquad ...(18.20)$$

in a recursive manner with respect $\grave{E}(S_2)$. The update of the parameters in the i^{th} node in layer L^{th} layer can be written as

$$\hat{\Theta}_i^L(k) = \hat{\Theta}_i^L(k-1) + \eta \frac{\partial E(k)}{\partial \hat{\Theta}_i^L(k)} \qquad ...(18.21)$$

where η is the learning rate and the gradient vector

$$\frac{\partial E}{\partial \hat{\Theta}_i^L} = \varepsilon_{L,i} \frac{\partial \hat{Z}_{L,i}}{\partial \hat{\Theta}_i^L} \qquad ...(18.22)$$

$\partial \hat{Z}_{L,i}$ being the node's output and $\varepsilon_{L,i}$, is the back propagated error signal.

18.3.3 Model of Combustion

The role of the combustion process is to produce the required heat energy for steam generation at the possible highest combustion efficiency. The efficiency depends on the completeness of burning and the waste heat taken away in the flue gas by the excess airflow. The higher the burning rate and smaller the waste heat is the higher efficiency. However, excess air is required for ensuring complete burning. The O_2 content of the flue gas is directly related to the amount of excess air. The aim of the combustion control, from the efficiency point of view, is to keep the O_2 content around 3-5% [16]. In multi-fuel fired fluidised bed power plants (Fig. 18.5), it is a difficult task due to the inhomogeneous properties of the fuel.

The combustion model, utilising the ANFIS structure based on [20], calculates the combustion power (Pcomb) and flue gas components (C_f), including the oxygen content, from the fuel screw *QHz*, signal primary airflow F_p, secondary airflow $F_{s.}$ (see Fig. 18.6)

The oxygen and combustion power controller (Fig. 18.6) consists of two parallel *PI* controllers. The error signal form the oxygen content drives the *PI* controller of the flue screw signal, while combustion power is controlled by the primary airflow. The structure of the *PI* controller is

$$U(s) = K_{pi} + K_h \frac{1}{S} \qquad i = 1, 2 \qquad ...(18.23)$$

The reference signals for the fuel screw *QHz*, primary airflow F_p and secondary airflow F_s signals are calculated by the linearization model as a function of the reference of the combustion power such as:

$$Q_{Hz} = 0.2662\mathrm{P}_{\mathrm{comb}} - 9.7207$$

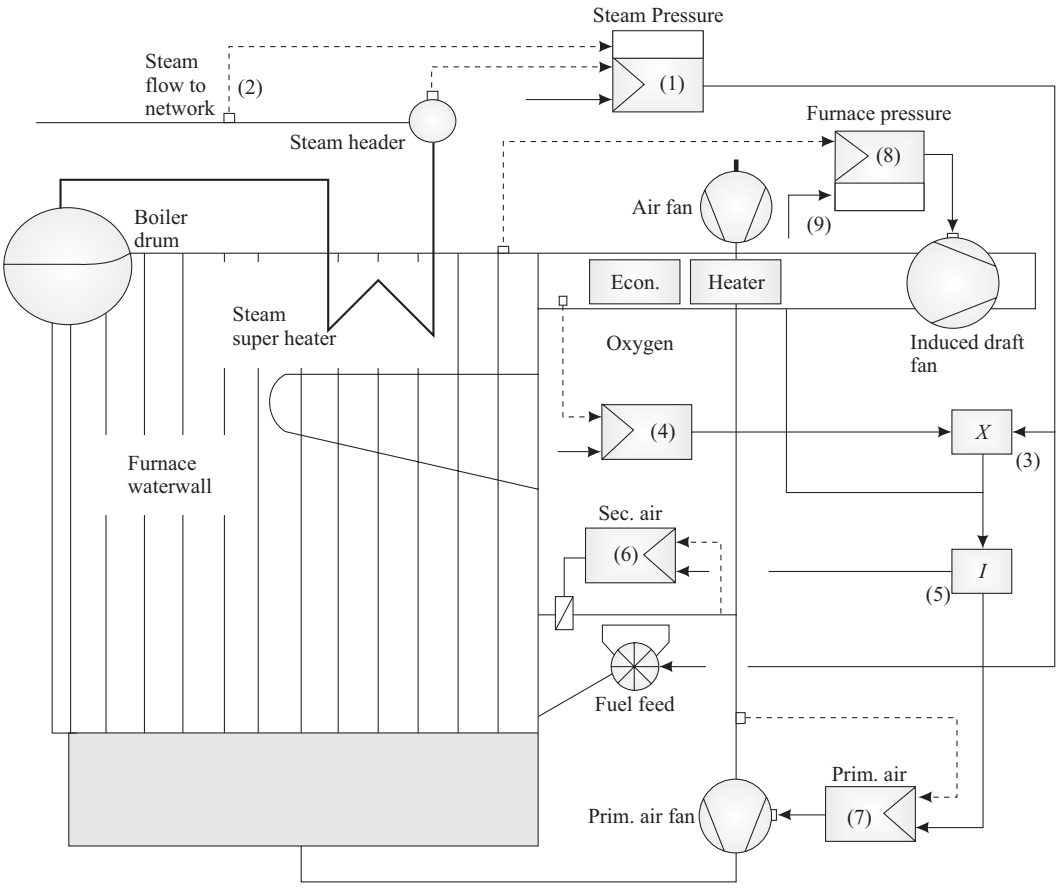

Fig. 18.5 Fluidized bed power plant.

$$F_p = 0.0737P_{\text{comb}} + 10.912 \qquad ...(18.24)$$

$$F_s = 0.2662P_{\text{comb}} - 4.005$$

18.3.4 Optimization of the PI-Controllers Using Genetic Algorithms

Standard genetic or genetic searching algorithms are used for numerical parameter optimization and are based on the principles of evolutionary genetics and the natural selection process [17].

A general genetic algorithm contains, usually, the next three procedures: selection, crossover and mutation. These procedures are responsible for the "global" search minimization function without testing all the solutions. Selection corresponds to keeping the best members of the population to the next generation to preserve the individual with good performance (elite individuals) in fitness function. Crossover originates new members for the population, by a process of mixing genetic information from both parents; depending of the selected parents the growing of the fitness of the population is faster or

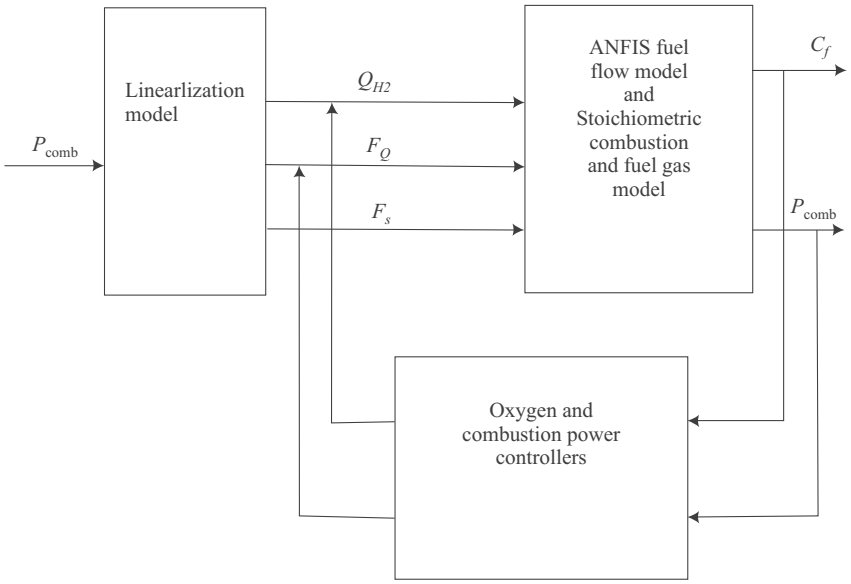

Fig. 18.6 Control system of combustion process.

lower. Among many other solutions, the parent selection can be done with the roulette method, by tournament, and random[18]. Mutation is a process by which a percentage of the genes are selected in a random fashion and changed. In the implemented algorithm a small population of 20 individuals, an elitism of 2 individuals was used, the crossover of one site splicing is performed and all the members are subjected to mutation except the elite. The mutation operator is a binary mask generated randomly according to a selected rate that is superposed to the existing binary codification of the population changing some of the bits [19]. Crossover is performed over half of the population, always including the elite. The individuals are randomly selected with equal opportunity to create the new population. The fitness function is

$$J = \frac{\sum_{1}^{N}(y_{\text{comb}} - \hat{y}_{\text{comb}})}{\sqrt{N}} + k \times \frac{\sum_{1}^{N}(y_{O2} - \hat{y}_{O2})}{\sqrt{N}} \qquad ...(18.25)$$

where the k is a weighting factor. In our case $k = 2$ to emphasize the importance of oxygen content which is directly related to the flue gas emissions.

The performance of the controller based on the ANFIS model is compared to the performance of the real process. The reference signal for the combustion power is taken from the measurement data. The simulation shows that by applying the new controller structure together with the ANFIS model, much smaller deviation in the oxygen content can be achieved while satisfying the same demand for combustion power.

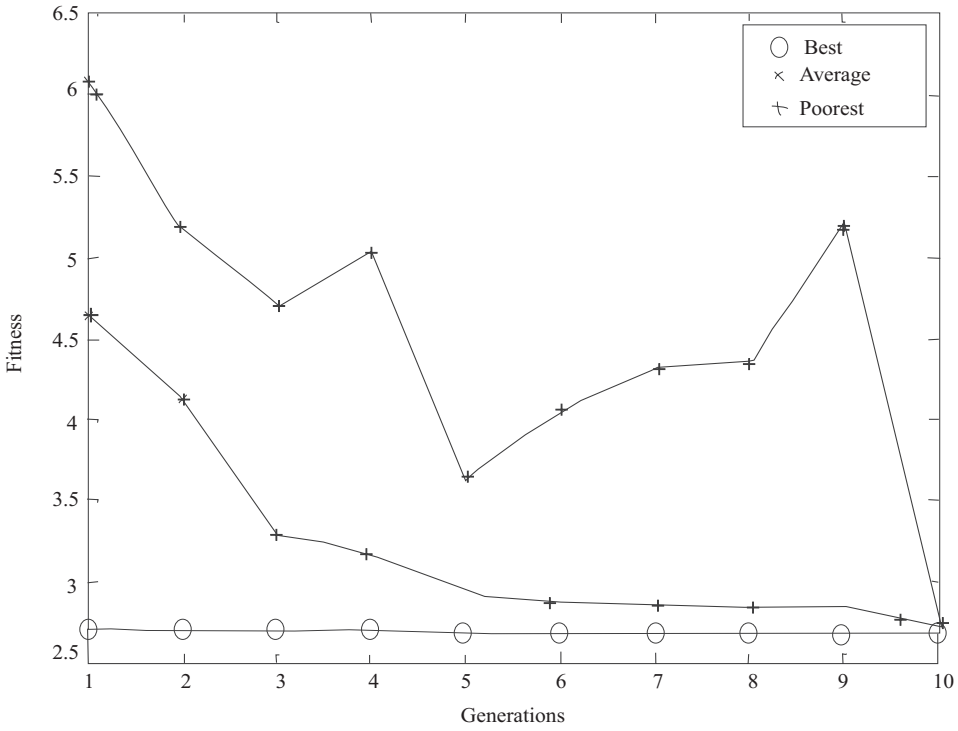

Fig. 18.7 Fitness function by the generation of the GA

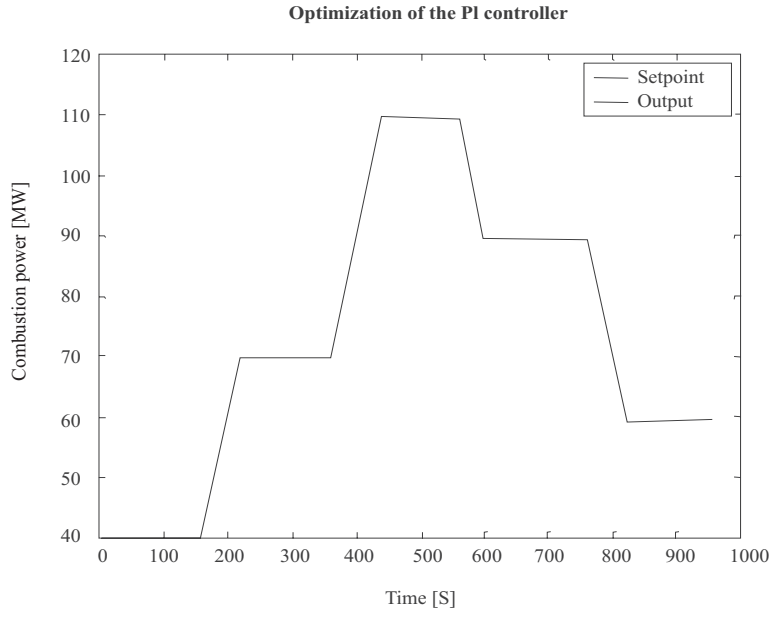

Fig. 18.8 PI combustion power controller optimization with GA.

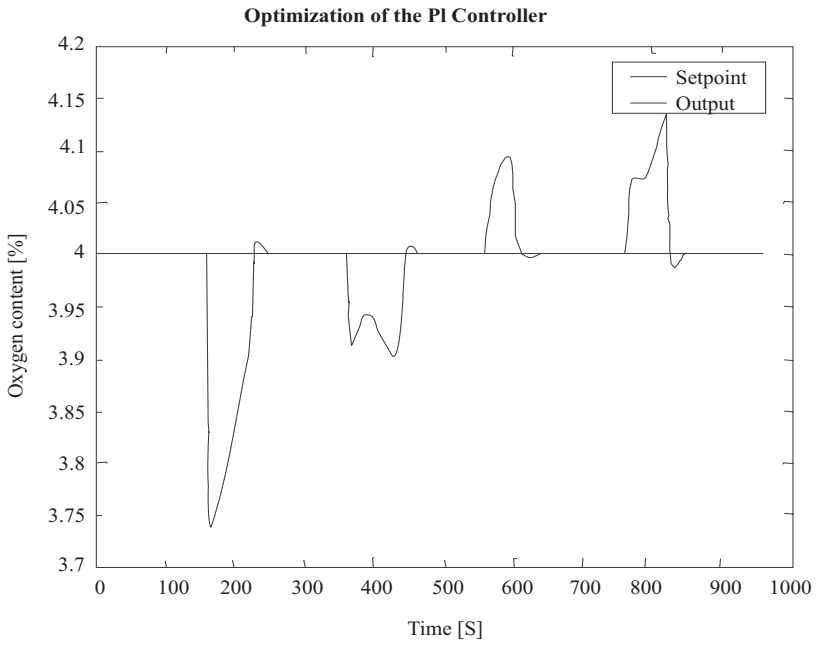

Fig.18.9 PI Oxygen content controller optimization.

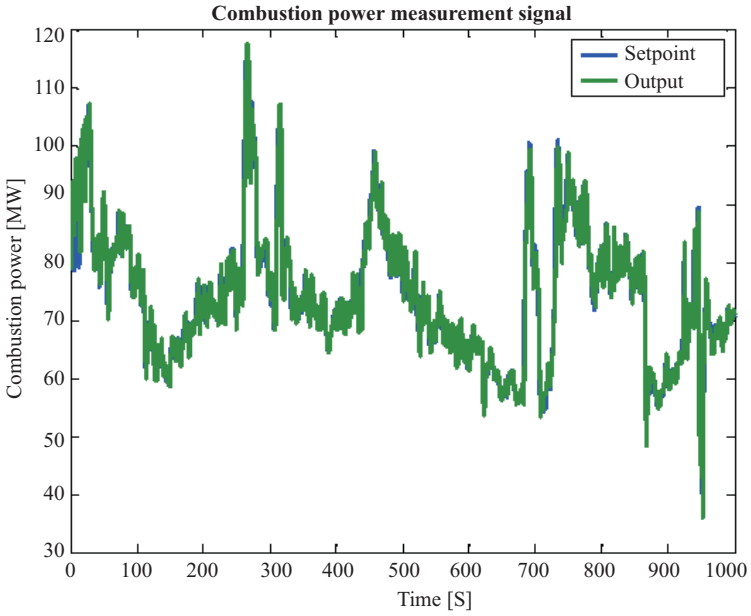

Fig. 18.10 Combustion power response: comparison of the achievement in real process and in the simulated control system.

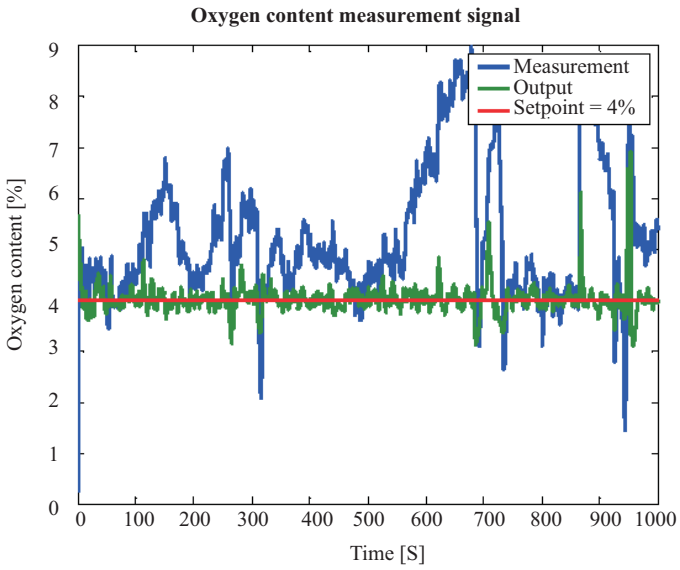

Fig. 18.11 Oxygen content response: comparison of the achievement in real process and in the simulated control system.

QUESTION BANK

1. Explain the application of hybrid fuzzy neural network for the tool breakage monitoring of end milling.
2. Explain the application of ANFIS to control the combustion process.

REFERENCES

1. Y. Altintas, I. Yellowley, and J. Tlusty, The Detection of Tool Breakage in Milling Operations, *Engineering for Industry,* Vol. 110, pp. 271-277, 1988.
2. J.C. Chen, A Fuzzy-Nets Tool-Breakage Detection System for End-Milling Operations, *International Journal of Advanced Manufacturing Technology*, Vol. 12, pp. 153-164, 1996.
3. J.C. Chen, and J.T. Black, J.T, A Fuzzy-Nets In-process (FNIP) System for Tool-Breakage Monitoring in End-Milling Operations, *International Journal of Machining*, *Tools manufacturing*, Vol. 37, No. 6, pp. 783-800, 1997.
4. J.S. Jang, C.T. Sun, and E. Mizutani, *Neuro-Fuzzy and Soft Computing*, New Jersey: Prentice Hall, 1997.
5. K. Jemielniak, Detection of Cutting Edge Breakage in Turning, *Annals of the CIRP*, Vol. 41, No. 1, pp. 97-100, 1992.
6. M. Lan, and Y. Naerheim, In-Process Detection of Tool Breakage in Milling, *Engineering for Industry, 108,* 191-197, 1986.

7. T.J. Ko, and D.W. Cho, D.W and M.Y. Jung, On-line Monitoring of Tool Breakage in Face Milling: Using a Self-Organized Neural Network, *Journal of Manufacturing Systems*, Vol. 14, No. 2, pp. 80-90, 1990.
8. I.N. Tansel, and C. McLanghlin, Detection of Tool Breakage in Milling Operation-I. The Time Series Analysis Approach, *International Journal of Machining*, *tools Manufacturing*, Vol. 33, No. 4, pp. 531-544, 1993.
9. I.N. Tansel, and C. McLanghlin, Detection of Tool Breakage in Milling Operation-II. The Neural Network Approach, *International Journal of Machining*, *Tools Manufacturing*, Vol.33, No. 4, pp. 545-588, 1993.
10. Y.S. Tarng, and B.Y. Lee, A Sensor for the Detection of Tool Breakage in NC Milling, *Journal of Materials Processing Technology*, Vol. 36, pp. 259-272, 1993.
11. D. Zhang, Y. Han, and D. Chen, On-line Detection of Tool Breakage Using Telemetering of Cutting Forces in Milling, *International Journal of Machining*, *Tools Manufacturing*, Vol. 35, No. 1, pp. 19-27, 1995.
12. R. Jang, C. Sun, and E. Mizutani, *Neuro-fuzzy and Soft Computation*, Prentice Hall, NJ,1997.
13. F.A. Alturki, and A.B. Abdennour, Neuro-fuzzy control of a steam boiler turbine unit, *Proceeding of the 1999 IEEE*, International Conference on Control Applications, Hawai, USA, pp. 1050-1055, 1999.
14. E. Ikonen, and K. Najim, Fuzzy neural networks and application to the FBC process, IEE Proc.-Control Theory Applications, Vol. 143, pp. 259-269, May 1996.
15. S.H. Kim, Y.H. Kim, K.B. Sim, and H.T Jeon, On Developing an adaptive neural-fuzzy control system, Proc. IEEE/RSJ Conference on intelligent robots and systems Yokohama, Japan, pp. 950-957, July 1993.
16. K. Leppäkoski and J. Kovacs, Hybrid model of oxygen content in flue gas. Proc. IASTED International Conference on Applied Modelling and Simulation, Nov, 2002, Cambridge, MA, USA, pp. 341- 346.
17. J.H. Holland, *Adaptation in Natural and Artificial System*, MIT Press, 1992.
18. J.S.Yang and M.L. West, A Case Study of PID Contoller tunning by Genetic Algorithm Proccedings of IASTED International Conference on Modelling and Control, Innsbruck, 2001.
19. J. Vieria, and A. Mota, Water Gas Heater Nonlinear Physical Model: Oprimazation with Genetic Algorithms Proccedings of IASTED International Conference on Modelling Identification and Control, Grindelwald, Switzerland 2004.
20. Z. Hímer, V. Wertz, J. Kovacs, and U. Kortela, Neuro- fuzzy model of flue gas oxygen content, Proceedings of IASTED International Conference on Modeling Identification and Control, Grindelwald, Switzerland, 2004.

Index

A

A back propagation neural network 235
α-Cut 11
Activation function 124
Adaline 125, 129, 133
Adaptive Critic 194
Adaptive Critic Element 192
Adaptive linear element 133
Adaptive Resonance Theory 182
Andlike 62
Andness is 62
ANFIS 229
ANFIS Structure 231
Anti-Reflexivity 20
Anti-Symmetricity 20
Applications of fuzzy logic 95
Approximate reasoning 41
Arithmetic mean 59
Arithmetical mean 58
ART 1 169, 183, 185
ART 2 211
Artificial network 122
Artificial neural 3
Artificial neural network 3, 123
ASE-ACE combination 192
Associative learning 125
Associative Memory 163
Associative Search 193
Associative Search Element 192
Associativity 54, 55
Asymmetric divergence 166
Auto-associator network 161
Auxiliary Hybrid Systems 218
Auxiliary hybrids 217
Average error 151
Averaging operators 58
Axon 3, 121

B

Back-propagation 3, 139, 142
Barto network 192
Barto's approach 192
Basic property 45
Bellman equations 196
Bias 127
Binary fuzzy relation 21
Biological neural network 121, 122
Boltzmann machines 165
Boolean logic 3

C

Calssical modus ponens 45
Cart-Pole system 194
Cartesian product 24
Cells 122
Center of-gravity method 74
Center-of-area 87
Centroid defuzzification 90, 9
Characteristic of fuzzy systems 9
Classical modus ponens 44
Classical modus tollens 45

Classical N-array relation 19
Clustering 169, 170
Committee 223
Committee of networks 223
Commutative 61
Commutativity 58
Compensatory 58
Competitive learning 170
Complement 17
Component extractor 181
Compositional rule 44
Conjugate gradient minimization 148
Conjunction rule 43
Contrast enhancement 187
Control of combustion 243
Control room temperature 100
Controller network 191
Convergence theorem 131
Convex fuzzy set 11
Cost function 172
Counter propagation 174
Critic 190

D

Defuzzification 67, 103
Defuzzifier 68, 86
Delta 131
Delta rule 134
Dendrites 3, 121
Dimensionality reduction 169
Direction set minimization method 148
Discrete membership function 10
Disjunction Rule 43
Dot product 170
Dubois and prade 55
Dynamic Programming 195
Dynamics 201

E

Eigenvectors 181
Elman Network 159
Embedded Hybrid Systems 218
Embedded hybrids 217
Empty Fuzzy Set 15
End-effector positioning 201
End milling 233
End milling cutting process 234
Entailment Rule 43
Entropy 63
Equivalence 20
Error back-propagation 222
Error function 173
Euclidean distance 171
Evaluation network 194
Exclusive-or 135
Expressive power 151
Extremal conditions 58

F

Feature extraction 169
Feed-forward network 139, 150
Feed-forward networks 125
Feedback control system 81
First-of-Maxima 87
Flexible manufacturing systems 233
FNN architecture 224
Follow-the-leader clustering algorithm 185
Forward Kinematics 200
Frank 55
Fuzzification 8
Fuzzy Approach 116
Fuzzy control 8
Fuzzy implication operator 32
Fuzzy implications 30
Fuzzy logic 1, 1, 6, 7
Fuzzy logic controller 82, 84
Fuzzy Logic Controllers 243
Fuzzy logic is 1, 1
Fuzzy neuron 220
Fuzzy Number 12
Fuzzy Point 15
Fuzzy relations 19, 21
Fuzzy rule-base system 71
Fuzzy set 9
Fuzzy singleton 63
Fuzzy systems 2

G

Gaussian membership functions 89
Generalised delta rule 140
Generalized Modus Ponens 44
Geometric mean 59
Godel implication 32
Graded response 164

Grading of apples 104
Gravity 87

H
Hamacher 55, 56
Harmonic mean 59
Hebbian learning 126
Hebbian Rule 180
Height defuzzification 88
Hidden Units 123, 153
Hopfield network 161
Human brain 3, 121
Hybrid fuzzy neural network 221
Hybrid neural network 220
Hybrid systems 217

I
Idempotency 58, 61
Inference mechanisms 72
Input units 123
Interpolation 42
Intersection 16, 21
Inverse Kinematics 200

J
Jordan Network 158

K
Kleene-Dienes implication 32
Kohonen network 177
Kullback information 166

L
Larsen inference Mechanism 77
Larsen system 66
Law of the excluded middle 2
Laws of Thought 2
Learning 127
Learning Rate 144
Learning Samples 152, 190
Least mean square 133, 135
Linear discriminant function 130
Linear threshold neural network 130
Linguistic variable 33, 34
Linguistic variable truth 35
LMS 131
LMS' rule 131
Local Minima 148
Long-term memory 183

M
Mamdani inference Mechanism 73
Mamdani system 66
Mamdani's implication operator 32, 63
Material implication 29
Mathematical neuron network 122
Max-Criterion 88
Maximum 56
Measure of dispersion 63
Median 59
Membership function 10
Middle-of-Maxima 87
Milling 210
Minimum 54
Modifiers 33
Momentum 144
Monotonicity 54, 55, 58, 61
Multi-layer network 140
Multi-layer perceptrons 137
Multi-input-multi-output 82

N
Negation rule 44
Network paralysis 148
Neural networks 2, 121
Neuro-fuzzy systems 8
Neuro-fuzzy-genetic systems 8
Neurofuzzy network 222
Neurons 3, 122, 164
Non-fuzzy approach 112
Normal fuzzy set 11
Normalization 186
Number of layers 127

O
Offset 127
One identy 54
Ordered weighted averaging 60
Original model 185
Orlike operator 62
Orness 62
Output units 123

P
Paradigms of learning 125
Partial order 20
Perceptron 2, 125, 129
Perceptron learning rule 131

Perceptron' learning rule 131
Perceptrons 2
Precisiated natural language 8
Precision 7
Principal component analysis 221
Principle of incompatibility 101
Principle of optimality 195
Probabilistic 56
Processing Units 123
Product 55
Product fuzzy conjunction 89
Product fuzzy implication 89
Projection 23
Projection Rule 44

Q
Q-learning 196
Quasi fuzzy number 12

R
Recurrent networks 125, 157
Reflexivity 20
Regular fuzzy neural network 220
Reinforcement learning 192
Reinforcement learning scheme 190
Representation 127
Road accidents 96
Robot arm dynamics 207
Robot control 200

S
Self-organization 125
Self-organizing networks 169
Semi-linear 124
Sequential hybrids 217
Sgn function 124
Shadow of fuzzy relation 24
Short-term memory 183
Sigmod 124
Significance 7
Simplified fuzzy Reasoning 77
Single layer feed-forward network 129
Single layer network 129, 130, 134
Singleton fuzzifier 89
Soft computing 8
Standard Strict 32
Stochastic function 193
Strong 56
Subset 46
Subsethood 14
Sugeno Inference Mechanism 75
Summed squared error 135
Sup-Min Composition 26
Superset 47
Supervised learning 125
Support 11
Symmetricity 20, 54
Symmetry 55

T
T 54, 56
t-conorm-based union 57
t-norm-based intersection 57
Taylor series 148
Test error rate 151
The linguistic variable truth 35
Threshold 127
Threshold (sgn) 130
Tool breakage 233
Total error 135
Total indeterminance 46
Total order 20
Traffic accidents and traffic safety 96
Trajectory generation 201
Transitivity 20
Translation rules 43
Trapezoidal fuzzy number 14
Triangular conorm 55
Triangular Fuzzy Number 13
Triangular norm 54
Tsukamoto inference mechanism 73
two layer feed-forward network 139
Two-input-single-output 82

U
Union 16, 22
Universal approximation theorem 142
Universal approximators 91
Universal fuzzy set 15
Unmanned flexible manufacturing system 233
Unsupervised learning 125

V
Vector quantisation 169, 174

W
Weak 55
Winner Selection 170

Y
Yager 55, 56

Z
Zero identity 55